WARSAW UNIVERSITY OF TECHNOLOGY
Faculty of Electronics and Information Technology

PRELIMINARY PROJECT

DATE : 22/04/2024

PREPARED BY

NAME : **İSMET ERDOĞAN**

NUMBER : **K-7721**

COURSE CODE : 103A-INXXX-ISP-EOOP
COURCE TITLE : OBJECT-ORIENTED PROGRAMMING
COURCE LECTURER : ROMAN PODRAZA

Contents

Description of the Project

Keyword: School

In this project, I was given "**School**" as the keyword. The classes that will be used in the project and the relationships between them were determined with the "**School**" keyword.

Classes and Relations Between Them

School is a comprehensive organization consisting of many bodies established for education and training purposes. The main organs of this organization are the "**Student**" receiving education, the "**Lecturer**" providing education, the "**Course**" as the education-training channel between the student and the lecturer, and the "**School**", which includes all these as well as carrying out administrative activities.

As a result, this organization will be represented by these 4 interconnected classes.

- School
- Lecturer
- Student
- Course

Class: School

The school keeps a record of enrolled students, lecturers and courses offered. School; can register or unregister students, hire or fire lecturers, create courses or delete existing courses, add or remove students enrolled in an existing course. The school can show all registered students, working lecturers and active courses.

Class: Lecturer

In this organization every lecturer is an instance of the "Lecturer" class. It stores various attributes such as the lecturer's name, title and branch. Lecturer, if a course does not have a Lecturer, Lecturer can be the Lecturer of that course. Or he/she can withdraw from the course for which he/she is currently a Lecturer. Additionally, the lecturer (instance) keeps a record of the courses he/she has given.

Class: Student

In this organization, each student is an instance of the "Student" class. It keeps attributes such as student name, student number (unique for each student), total credits of courses taken. The student can register or unregister from the school. The student can enroll in a course or unenroll his/her registration from that course and find out the total credits he/she has received during the semester. Additionally, student (instance) keeps a record of courses they are registered for.

Class: Course

In this organization, each course is an instance of the "Course" class. Course encapsulates attributes such as course name, code, number of course credits. Course can assign a Lecturer, assign another Lecturer to replace the existing Lecturer, or terminate the Lecturer from his or her teaching duties (not from his or her position at the school). Course can add or remove students from the relevant course. Can change the number of credits of the course. Additionally, course (instance) keeps a record of lecturer (as a Lecturer pointer) teaching the course and students taking the course (as a singly-linked list which points students).

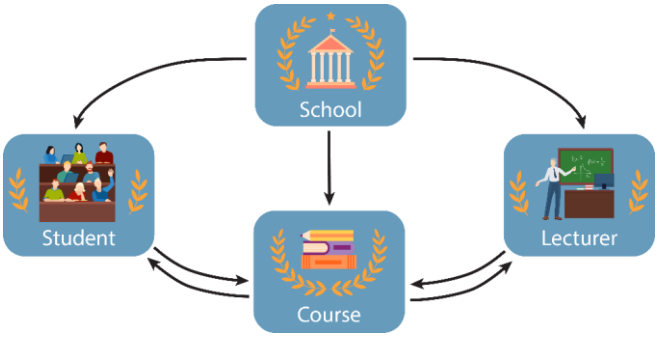

Figure 1: Classes and Relationship Between Them

Limits, Restrictions and Assumptions

Limits

1. Each lecturer can give a maximum of 3 courses.
2. There can be a maximum of 10 students enrolled in each course.
3. Each course can have only 1 lecturer.
4. A course can have a minimum of 1 credit and a maximum of 4 credits.

Restrictions

1. Student number is unique and cannot be changed later.
2. A second course cannot be created with the same course name or course code.

Assumptions

1. There may be no students registered for the course.
2. If a teacher teaching a course is fired (and the course instructor has not been updated previously), the students involved will be removed from the course and the course will disappear.
3. If the student is unregistered from the school, his enrolments will also be deleted from all courses in which he is enrolled.
4. If a course is removed by the school, the student's course records and the relationship between the lecturer and the course are also deleted.

Case Study

Memory Map

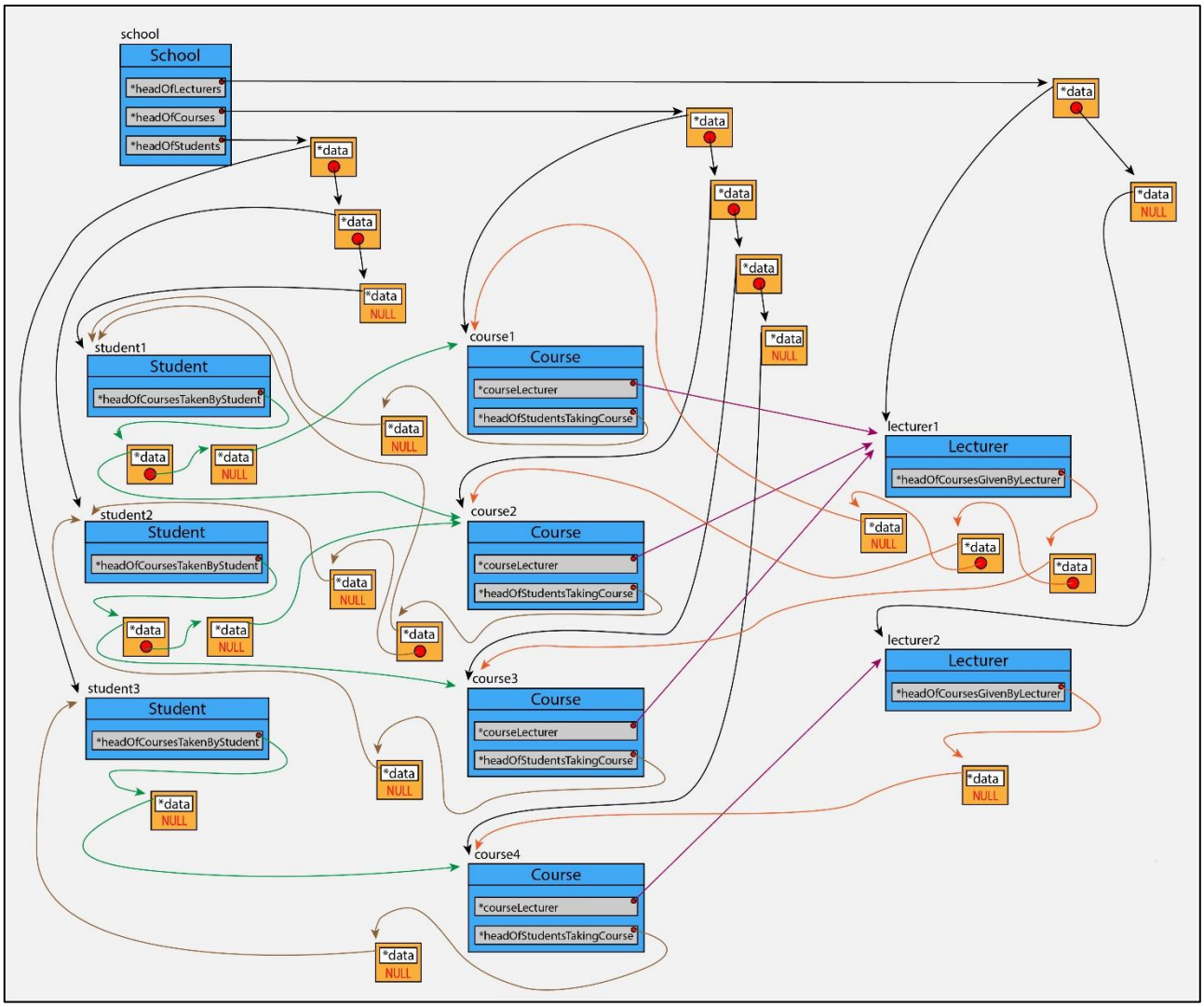

Figure 2: Memory Map

Explanation of Memory Map

Figure 2 shows the instances of schools, students, lecturers and course classes in the organization and the relationship between them.

Blue boxes represent instances. The part written in large font in the box indicates which class the object is. The text outside and just above the blue boxes shows the name of the object.

The gray boxes within the blue boxes represent the pointer and show the first element of the single-linked list (the pointer in the Course class does not show this because it points directly to the object since each course will have a single lecturer).

Yellow boxes indicate single-linked list elements. The white box inside represents the pointer named *data and points to the object. The red circle inside represents the pointer named *next and shows the next element. If there is no element after that element, NULL appears instead of the red circle.

As seen in Figure 2, there is 1 School, 3 Student, 4 Course, and 2 Lecture instances.

All of them are included in the singly-linked lists pointed by the pointers in the School.

student1 took 2, student2 2 and student3 1 course. The pointer belonging to the instance points to the beginning of the single linked list that points to the course taken by these students.

1 student took course1, 2 students took course2, 1 student took course3 and 1 student took course4, and the pointer named *headOfStudentsTakingCourse in the instance is the pointer that points to the head of the single-linked list that points to the student taking the course.

The pointer named *courseLecturer in the Course points directly to the lecturer without using a singly-linked list, unlike other pointers. Because only one teacher can teach each course.

lecturer1 teaches course1, course2 and course3. lecturer2 only teaches course4.

Declaration of the Classes

School.h

```
// School.h

#ifndef SCHOOL_H
#define SCHOOL_H

// forward declarations
class Lecturer;
class Course;
class Student;


class School {
//    overloading Stream Insertion Operator (<<)
//    # Function: when it is called, print all details of the School
    friend std::ostream& operator<<(std::ostream& output, School&
school);
public:
//    Constructor
//    # Parameters
//      - name: name of the School ("Warsaw University of
Technology")
    School(const std::string &name);
//    Destructor
//    # Function
//      -> With dynamic memory management, it frees the Student,
Course and Lecturer sections that it has reserved
//       (elements of single-linked list).
    ~School();

//    hireLecturer
//    # Parameters
//      - lecturer: Teacher object to be hired
//    # Function
//      -> hire a new Lecturer at the School (adds to Lecturer list)
//    # Return
//      - true: If the hiring process is successful
//      - false: If the hiring process is unsuccessful
//          -> The Lecturer may already be hired
//          -> Dynamic Memory problem...
    bool hireLecturer(Lecturer& lecturer);
//    fireLecturer
//    # Parameters
//      - lecturer: Teacher object to be fired
```

```
//     # Function
//       -> fire a Lecturer from the School (removes from offered
Course list)
//     # Return
//       - true: If the firing process is successful
//       - false: If the firing process is unsuccessful
//           -> The Lecturer may already be fired or has no job...
    bool fireLecturer(Lecturer& lecturer);
//     fireLecturerByName
//     # Parameters
//       - name: name of the Teacher object to be fired
//     # Function
//       -> fire a Lecturer from the School (removes from offered
Course list)
//     # Return
//       - true: If the firing process is successful
//       - false: If the firing process is unsuccessful
//           -> If there is no Lecturer corresponding to the entered
name among
//           the working Lecturers...
    bool fireLecturerByName(const std::string &name);

//     addCourse
//     # Parameters
//       - course: Course object to be opened at school
//       - courseLecturer: Lecturer who will give the Course
//     # Function
//       -> Opens a new Course at the School that Student(s) can take.
(adds to
//       offered Course list)
//     # Return
//       - true: If the adding Course is successful
//       - false: If the adding Course is unsuccessful
//           -> Lecturer may not be working at school
//           -> Lecturer may have reached the maximum number of
lessons he/she can teach
//           -> Course may have already been added...
    bool addCourse(Course& course, Lecturer& courseLecturer);
//     removeCourse
//     # Parameters
//       - course: Course object to be removed at School
//     # Function
//       -> Removes an existing Course at the School (removes from
offered Course list)
//     # Return
//       - true: If the removing Course is successful
//       - false: If the removing Course is unsuccessful
//           -> The Course may not have been offered at the school...
```

```
    bool removeCourse(Course& course);
//     removeCourse
//     # Parameters
//       - code: Course code of the course to be removed at School
(like ECULT)
//     # Function
//       -> Removes an existing Course at the School (removes from
offered Course list)
//     # Return
//       - true: If the removing Course is successful
//       - false: If the removing Course is unsuccessful
//           -> There may not be a course corresponding to the entered
code among the courses
//           offered at the school...
    bool removeCourseByCode(const std::string &code);

//     registerStudent
//     # Parameters
//       - student: Student object to be registered to school
//     # Function
//       -> register the Student in school (adds to registered Student
list)
//     # Return
//       - true: If the registering Student is successful
//       - false: If the registering Student is unsuccessful
//           -> The Student may already be registered
//           -> The student's number may be the number of another
Student currently
//           registered in the school...
    bool registerStudent(Student& student);
//     unregisterStudent
//     # Parameters
//       - student: Student object to be unregistered to school
//     # Function
//       -> unregister the Student in school (removes from registered
Student list)
//     # Return
//       - true: If the unregistering Student is successful
//       - false: If the unregistering Student is unsuccessful
//           -> The Student may not be registered in School...
    bool unregisterStudent(Student& student);
//     unregisterStudentByStudentNumber
//     # Parameters
//       - studentNumber: student number of Student object to be
unregistered to school
//     # Function
//       -> unregister the Student in school (removes from registered
Student list)
```

```
//     # Return
//       - true: If the unregistering Student is successful
//       - false: If the unregistering Student is unsuccessful
//           -> There may not be Student registered to the entered
student number...
    bool unregisterStudentByStudentNumber(const std::string
&studentNumber);

//     addStudentToCourse
//     # Parameters
//       - student: Student object to add to Course
//       - course: Course object to which the student will be added
//     # Function
//       -> Adds the student to the course. In this way, the Student
//       is enrolled for the course. It also establishes relationships
between
//       the student and the course.
//     # Return
//       - true: If the adding Student to Course is successful
//       - false: If the adding Student to Course is unsuccessful
//           -> The Student may not be enrolled in school
//           -> The Course may not be open (such a Course may not be
offered by
//           the school)
//           -> The Course may be at capacity
//           -> The Student may already be enrolled in the course...
    bool addStudentToCourse(Student& student, Course& course);
//     removeStudentFromCourse
//     # Parameters
//       - student: Student object to remove from Course
//       - course: Course object to which the student will be removed
//     # Function
//       -> Removes the student from the Course. In this way, the
Student
//       is unEnrolled for the course.
//     # Return
//       - true: If the removing Student from the Course is successful
//       - false: If the removing Student from the Course is
unsuccessful
//           -> The Student may not be enrolled in school
//           -> The Course may not be open (such a Course may not be
offered by
//           the school)
//           -> The Student may already be unEnrolled in the course...
    bool removeStudentFromCourse(Student& student, Course& course);

//     printLecturers
//     # Function
```

```
//      -> Prints all Lecturer(s) hired at the School.
    void printLecturers() const;
//    printCourses
//    # Function
//      -> Prints all Course(s) offered by the School.
    void printCourses() const;
//    printStudents
//    # Function
//      -> Prints all Student(s) registered in the School.
    void printStudents() const;

//    Getter functions for private data members
    std::string getName() const;
    int getNumberOfLecturers() const;
    int getNumberOfCourses() const;
    int getNumberOfStudents() const;
private:
//    name: Corresponds to the School Name (e.g. Istanbul Technical
University)
    std::string name;

//    element of singly-linked list which keeps hired Lecturer(s)
    struct lecturerElement {
        Lecturer* data;
        lecturerElement *next;
    };
    lecturerElement *headOfLecturers;
//    number of hired Lecturer(s)
    int numberOfLecturers;

//    element of singly-linked list which keeps Course(s) offered by
the School
    struct courseElement {
        Course* data;
        courseElement *next;
    };
    courseElement *headOfCourses;
//    number of Course(s) offered by the School
    int numberOfCourses;

//    element of singly-linked list which keeps Student(s) registered
in School
    struct studentElement {
        Student* data;
        studentElement *next;
    };
//    number of Student(s) registered in School
    studentElement *headOfStudents;
```

```
    int numberOfStudents;

//    private methods
//    -> These methods were declared because they will be needed in
public methods
//    due to their various functionalities.
//    -> These methods search various single-linked lists maintained
by the object.
//    -> Returns TRUE if the searched object is in the current list,
FALSE otherwise.
//    -> Since the current and previous parameters are reference to
pointers,
//    they are useful for removing elements from the list.
    bool findLecturer(const Lecturer& lecturer, lecturerElement*&
current,
                      lecturerElement*& previous) const;
    bool findLecturerByName(const std::string &name,
lecturerElement*& current,
                      lecturerElement*& previous) const;
    bool findCourse(const Course& course, courseElement*& current,
                    courseElement*& previous) const;
    bool findCourseByCode(const std::string &code, courseElement*&
current,
                    courseElement*& previous) const;
    bool findStudent(const Student& student, studentElement*&
current,
                     studentElement*& previous) const;
    bool findStudentByStudentNumber(const std::string &studentNumber,
studentElement*& current,
                     studentElement*& previous) const;
};


#endif //SCHOOL_H
```

Lecturer.h

```
// Lecturer.h

#ifndef LECTURER_H
#define LECTURER_H

// forward declaration
class Course;


class Lecturer {
//    overloading Stream Insertion Operator (<<)
//    # Function: when it is called, print all details of the
Lecturer
    friend std::ostream& operator<<(std::ostream& output, Lecturer&
lecturer);
public:
//    Constructor
//    # Parameters
//      - name: name of the Lecturer
//      - title: title of the Lecturer
//      - branch: branch of the Lecturer
    Lecturer(const std::string &name, const std::string &title, const
std::string &branch);
//    Destructor
//    # Function
//      -> With dynamic memory management, it frees Course sections
that it has reserved (elements
//      of single-linked list).
    ~Lecturer();

//    becomeLecturerOfCourse
//    # Parameters
//      - course: Course object where Lecturer will teach
//    # Function
//      -> The Lecturer becomes the Lecturer of the course. The
Course is added to the list
//      of Course(s) given by the Lecturer, It also establishes
relationships between the Lecturer
//      and the Course (e.g. due to the relationship, the
courseLecturer pointer of the Course
//      now points to this Lecturer object).
//    # Return
//      - true: If the process of becoming the Lecturer of the Course
is successful
//      - false: If the process of becoming the Lecturer of the
Course is unsuccessful
```

```
//          -> The Lecturer object may not be employed by the School
//          -> The Course may not be offered by the School
//          -> The Course may already have a Lecturer
//          -> The Lecturer may have reached the maximum number of
courses they can offer...
    bool becomeLecturerOfCourse(Course& course);
//    quitTeachingTheCourse
//    # Parameters
//      - course: Course object where Lecturer will quit teaching the
Course
//    # Function
//      -> The Lecturer exits the Course in which he or she is
currently teaching. Previously
//      established connections between Lecturer and Course are also
broken at this stage.
//    # Return
//      - true: If the process of quiting teaching the Course is
successful
//      - false: If the process of quiting teaching the Course is
unsuccessful
//          -> The Lecturer may not currently be teaching this
course...
    bool quitTeachingTheCourse(Course& course);

//    isGivenCourseCapacityFull
//    # Function
//      -> Returns whether the Lecturer has reached the maximum
number of courses
//      Lecturer can teach
//    # Return
//      - true: If the number of Courses the Lecturer can give is
maximum
//      - false: If the number of Courses the Lecturer can give is
not maximum
    bool isGivenCourseCapacityFull() const;

//    printCourses
//    # Function
//      -> Prints all Course(s) teaching by the Lecturer
    void printCourses() const;

//    Getter functions for private data members
    std::string getName() const;
    std::string getTitle() const;
    std::string getBranch() const;
    int getNumberOfCoursesGivenByLecturer() const;
    bool getJobStatus() const;
private:
```

```
//    name: Corresponds to the name of the Lecturer (e.g. Isaac
Johnson)
    std::string name;
//    title: Corresponds to the title of the Lecturer (e.g.
Professor)
    std::string title;
//    branch: Corresponds to the branch of the Lecturer (e.g.
Automatic Control)
    std::string branch;
//    jobStatus: It is a variable that indicates whether the Lecturer
is currently working
//    at the school or not.
//      - true: working at school
//      - false: not working at school
    bool jobStatus;

//    element of singly-linked list which keeps Course(s) given by
the Lecturer
    struct courseElement {
        Course* data;
        courseElement *next;
    };
    courseElement *headOfCoursesGivenByLecturer;
//    number of Course(s) given by the Lecturer
    int numberOfCoursesGivenByLecturer;

//    private methods
//    -> These method(s) were declared because they will be needed in
public methods
//    due to their various functionalities.
//    -> These method(s) search various single-linked lists
maintained by the object.
//    -> Returns TRUE if the searched object is in the current list,
FALSE otherwise.
//    -> Since the current and previous parameters are reference to
pointers,
//    they are useful for removing elements from the list.
    bool findCourse(const Course& course, courseElement*& current,
                    courseElement*& previous) const;
};


#endif //LECTURER_H
```

Course.h

```
// Course.h

#ifndef COURSE_H
#define COURSE_H

// forward declarations
class Lecturer;
class Student;


class Course {
//    overloading Stream Insertion Operator (<<)
//    # Function: when it is called, print all details of the Course
    friend std::ostream& operator<<(std::ostream& output, Course&
course);
public:
//    Constructor
//    # Parameters
//      - name: name of the Course
//      - code: code of the Course
//      - numberOfCredits: number of credits of Course
    Course(const std::string& name, const std::string& code, int
numberOfCredits);
//    Destructor
//    # Function
//      -> With dynamic memory management, it frees Student(s)
sections that it has reserved (elements
//      of single-linked list).
    ~Course();

//    assignLecturer
//    # Parameters
//      - lecturer: Lecturer object to be assigned to the Course
//    # Function
//      -> The Lecturer object is assigned to the course as the
course Lecturer,
//      that is, it becomes the Lecturer of the relevant course. It
also establishes relationships
//      between the Course and the Lecturer. (e.g. due to the
relationship, The Course object is
//      added to the list of courses offered by the relevant
Lecturer. Additionally, the
//      courseLecturer pointer in the Course points to this Lecturer
object.)
//    # Return
//      - true: If the process of assigning the Lecturer to the
```

```
Course is successful
//      - false: If the process of assigning the Lecturer to the
Course is unsuccessful
//          -> The Course may not have been opened/offered by the
school (status=false)
//          -> The Course may already have a Lecturer.
//          -> The Lecturer object may not be employed by the School
//          -> The Lecturer to be assigned may have reached the
maximum number of Courses
//          he/she can offer...
    bool assignLecturer(Lecturer& lecturer);
//    updateLecturer
//    # Parameters
//      - lecturer: Lecturer object to be reassigned to the Course
//    # Function
//      -> The Lecturer object is reassigned to the course as the
course Lecturer. Previous
//      Course-Lecturer connections are broken and a new one is
established.
//    # Return
//      - true: If the process of updating the Lecturer to the Course
is successful
//      - false: If the process of updating the Lecturer to the
Course is unsuccessful
//          -> The Course may not have been opened/offered by the
school (status=false)
//          -> The Lecturer object may not be employed by the School
//          -> The Lecturer to be reassigned may have reached the
maximum number of Courses
//          he/she can offer...
    bool updateLecturer(Lecturer& lecturer);
//    fireLecturerFromCourse
//    # Parameters
//      - lecturer: Lecturer object to be fired from the Course
//    # Function
//      -> The Lecturer object is removed from the Course and any
previously
//      established connections are broken.
//    # Return
//      - true: If the process of firing the Lecturer from the Course
is successful
//      - false: If the process of firing the Lecturer from the
Course is unsuccessful
//          -> The Course may not have been opened/offered by the
school (status=false)
//          -> The course may not currently have a Lecturer. This
means that
//          after this process, the Course will not have a Lecturer
```

```
(until it
//          is assigned again)...
    bool fireLecturerFromCourse();

//    addStudent
//    # Parameters
//      - student: Student object to add to Course
//    # Function
//      -> Adds the Student entered as an argument to this method to
the relevant course
//      (this). While doing this, it establishes connections between
Course-Student
//      (such as adding this student to the list of students taking
the course,
//      adding this course to the list of courses taken by the
student).
//    # Return
//      - true: If the process of adding the Student to the Course is
successful
//      - false: If the process of adding the Student to the Course
is unsuccessful
//          -> The Course may not have been opened/offered by the
school (status=false)
//          -> Student may not be registered in the School
//          -> The Student may already be enrolled in the course
//          -> The course may be at capacity...
    bool addStudent(Student& student);
//    removeStudent
//    # Parameters
//      - student: Student object to remove from Course
//    # Function
//      -> Remove the Student entered as an argument to this method
from the relevant
//      course (this). While doing this, Previous connections between
Student and
//      Course are broken.
//    # Return
//      - true: If the process of removing the Student from the
Course is successful
//      - false: If the process of removing the Student from the
Course is unsuccessful
//          -> The Student may not be enrolled in school
//          -> The Course may not have been opened/offered by the
school (status=false)
//          -> The student may not be currently taking this course...
    bool removeStudent(Student& student);

//    changeNumberOfCredits
```

```
//     # Parameters
//       - newNumberOfCredits: new number of credits of the course
//     # Function
//       -> Updates the number of credits of the course
//     # Return
//       - true: If the process of updating the number of credits of
the Course is successful
//       - false: If the process of updating the number of credits of
the Course is unsuccessful
//           -> The number of new credits entered may not be within
the appropriate range...
    bool changeNumberOfCredits(int newNumberOfCredits);
//     isThereACourseLecturer
//     # Function
//       -> Returns whether the Course currently has a Lecturer.
//     # Return
//       - true: If the Course has a Lecturer
//       - false: If the Course has not a Lecturer
    bool isThereACourseLecturer() const;
//     isCourseCapacityFull
//     # Function
//       -> Returns whether the Course is at full capacity
//     # Return
//       - true: If the Course is at full capacity
//       - false: If the Course is not at full capacity
    bool isCourseCapacityFull() const;
//     printStudents
//     # Function
//       -> Prints all Student(s) enrolled in the Course
    void printStudents() const;

//     Getter functions for private data members
    std::string getName() const;
    std::string getNameOfTheCourseLecturer() const;
    int getNumberOfStudentsTakingCourse() const;
    int getNumberOfCredits() const;
    bool getCourseStatus() const;
private:
//     name: Corresponds to the name of the Course (e.g. Object-
Oriented Programming)
    std::string name;
//     code: Corresponds to the code of the Course (e.g. EOOP)
    std::string code;
//     numberOfCredits: Corresponds to the number of credits of the
course like ECTS (e.g. 4 ECTS)
    int numberOfCredits;
//     courseStatus: It is the variable that indicates whether the
course is active or not.
```

```
//      - true: The course is offered by the school, so students can
enroll, lecturer
//      can become of the lecturer of the course
//      - false: The course is offered by the school, so students
cannot enroll, lecturer
//      can not become of the lecturer of the course
    bool courseStatus;

//    courseLecturer: Points to the Lecturer of the course, otherwise
it is nullptr.
    Lecturer* courseLecturer;

//    element of singly-linked list which keeps Student(s) taken this
Course
    struct studentElement {
        Student* data;
        studentElement *next;
    };
    studentElement *headOfStudentsTakingCourse;
//    number of Student(s) taken this Course
    int numberOfStudentsTakingCourse;

//    private methods
//    -> These method(s) were declared because they will be needed in
public methods
//    due to their various functionalities.
//    -> These method(s) search various single-linked lists
maintained by the object.
//    -> Returns TRUE if the searched object is in the current list,
FALSE otherwise.
//    -> Since the current and previous parameters are reference to
pointers,
//    they are useful for removing elements from the list.
    bool findStudent(const Student& student, studentElement*&
current,
                     studentElement*& previous) const;
};


#endif //COURSE_H
```

Student.h

```
// Student.h

#ifndef STUDENT_H
#define STUDENT_H

// forward declarations
class School;
class Course;


class Student {
//    overloading Stream Insertion Operator (<<)
//    # Function: when it is called, print all details of the Student
    friend std::ostream& operator<<(std::ostream& output, Student&
student);
public:
//    Constructor
//    # Parameters
//      - name: name of the Student
//      - studentNumber: student number of the Student
    Student(const std::string &name, const std::string
&studentNumber);
//    Destructor
//    # Function
//      -> With dynamic memory management, it frees Course(s)
sections that it has reserved (elements
//      of single-linked list).
    ~Student();

//    registerToSchool
//    # Parameters
//      - school: The School object where the student will register
//    # Function
//      -> Registers the Student (this) in School (argument). At this
time, the connection between
//      the student and the school is established (this Student is
also added to the list of
//      Student(s) registered at the School).
//    # Return
//      - true: If the process of registering the Student in School
is successful
//      - false: If the process of registering the Student in School
is unsuccessful
//          -> The Student may be currently enrolled in the School
//          -> The student number may be used by another student
registered at the school...
```

```
    bool registerToSchool(School& school);
//    unregisterFromSchool
//    # Function
//      -> Unregisters the Student (this) from School. At this time,
the connection between
//      the student and the school is broken.
//    # Return
//      - true: If the process of unregistering the Student in School
is successful
//      - false: If the process of unregistering the Student in
School is unsuccessful
//          -> The Student may not be currently enrolled in the
School...
    bool unregisterFromSchool();

//    enrollCourse
//    # Parameters
//      - course: Course object in which the Student will enroll
//    # Function
//      -> The Student enroll the Course and becomes one of the
Student(s) of the Course. At this
//      time, connections between Student and Course are established
(The Course object is added
//      to the list of courses taken by the Student. The Student
object is added to the list of
//      students taking the Course.).
//    # Return
//      - true: If the process of enrolling the Student for the
Course is successful
//      - false: If the process of enrolling the Student for the
Course is unsuccessful
//          -> The Student may not be currently enrolled in the
School
//          -> The Student may already be a Student of the Course
//          -> The course may be at capacity
//          -> The Course may not be active (The School may not have
added the Course
//          or the Course may not have a Lecturer)...
    bool enrollCourse(Course& course);
//    unEnrollCourse
//    # Parameters
//      - course: Course object in which the Student will unEnroll
//    # Function
//      -> The Student unEnroll the Course, so is removed from the
Course. At this
//      time, connections between Student and Course are broken.
//    # Return
//      - true: If the process of unEnrolling the Student for the
```

```
Course is successful
//      - false: If the process of unEnrolling the Student for the
Course is unsuccessful
//          -> The Student may not be currently enrolled in the
School
//          -> The Course may not be active
//          -> The Student may not be taking this Course...
    bool unEnrollCourse(Course& course);

//    printCourses
//    # Function
//      -> Prints all Course(s) taken by the Student
    void printCourses() const;

//    Getter functions for private data members
    std::string getName() const;
    std::string getStudentNumber() const;
    int getTotalCreditsOfCoursesTaken() const;
    bool getCurrentActivityStatus() const;
private:
//    name: Corresponds to the name of the Student (e.g. Rudolf
Kalman)
    std::string name;
//    studentNumber: Corresponds to the student number of the Student
(e.g. 07800235)
    std::string studentNumber;
//    totalCreditsOfCoursesTaken: Corresponds to the sum of the
number of credits of Course(s)
//      taken by the Student.
    int totalCreditsOfCoursesTaken;
//    activeStudent: It is the variable that indicates whether the
student is registered in School or not.
//      - true: Student is registered in School.
//      - false: Student is not registered in School.
    bool activeStudent;

//    element of singly-linked list which keeps Course(s) taken by
(this) Student
    struct courseElement {
        Course* data;
        courseElement *next;
    };
    courseElement *headOfCoursesTakenByStudent;
//    number of Course(s) taken by (this) Student
    int numberOfCoursesTakenByStudent;

//    private methods
//    -> These method(s) were declared because they will be needed in
```

```
public methods
//    due to their various functionalities.
//    -> These method(s) search various single-linked lists
maintained by the object.
//    -> Returns TRUE if the searched object is in the current list,
FALSE otherwise.
//    -> Since the current and previous parameters are reference to
pointers,
//    they are useful for removing elements from the list.
    bool findCourse(const Course& course, courseElement*& current,
                    courseElement*& previous) const;
};


#endif //STUDENT_H
```

Testing

Case 1: Hiring a New Lecturer

```
//    ##################################################################
//    Case 1: Hiring a New Lecturer
//    ------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");

    bool test1 = school.hireLecturer(lecturer);
//    test1 -> true (expected)
    if (test1)
//        printLecturers() is called to see if Lecturer is hired
        school.printLecturers();

//    Expected Result: hireLecturer performs the hire operation and
//    returns true.  Afterward, the condition required to enter the
//    if block is met and Lecturer is printed on the screen.
////
##################################################################
```

Case 2: Hiring a New Lecturer (already existing)

```
//    ##################################################################
//    Case 2: Hiring a New Lecturer (already existing)
//    ------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");

    school.hireLecturer(lecturer);

    bool test2 = school.hireLecturer(lecturer);
//    test2 -> false (expected)
    if (!test2)
//        printLecturers() is called to see if second same Lecturer
//        is not hired
        school.printLecturers();
```

```
//    Expected Result: The hireLecturer() method prints an error
//    message and returns false. As a result, a single Lecturer
//    appears in the output.
//    ##################################################################
```

Case 3: Firing the Lecturer

```
//    ##################################################################
//    Case 3: Firing the Lecturer
//    ------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");

    school.hireLecturer(lecturer);
//        printLecturers() is called to see if Lecturer is hired
    school.printLecturers();

    bool test3 = school.fireLecturer(lecturer);
//    test3 -> true (expected)
    if (test3)
//        printLecturers() is called to see if Lecturer is fired
        school.printLecturers();

//    Expected Result: The fireLecturer method performs the fire
//    operation and returns true. As a result, Lecturer does not
//    appear in the printLecturers method in the if block.
//    ##################################################################
```

Case 4: Firing the Lecturer (non-existing)

```
//    ##################################################################
//    Case 4: Firing the Lecturer (non-existing)
//    ------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");

    bool test4 = school.fireLecturer(lecturer);
//    test4 -> false (expected)
    if (!test4)
//        printLecturers() is called to see nothing
        school.printLecturers();
```

```
//    Expected Result: The fireLecturer method cannot perform the
//    fire operation because the Lecturer has not been hired
//    before. It returns false as a result. And when the hired
//    teachers are printed on the screen, nothing appears.
//    ###############################################################
```

Case 5: Adding a New Course

```
//    ###############################################################
//    Case 5: Adding a New Course
//    ---------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);

    school.hireLecturer(lecturer);

    bool test5 = school.addCourse(course, lecturer);
//    test5 -> true (expected)
    if (test5)
//        printCourses() is called to see if Course is added
        school.printCourses();

//    Expected Result: The addCourse method performs the function of
//    opening a course and returns true. The course is then printed
//    on the screen.
//    ###############################################################
```

Case 6: Adding a New Course (with same code or same name)

```
//    ###############################################################
//    Case 6: Adding a New Course (with same code or same name)
//    ---------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);
    Course course2("Advanced Topics in Building Iron Man", "IBIM",
4);
```

```
    school.hireLecturer(lecturer);
    school.addCourse(course, lecturer);

    bool test6 = school.addCourse(course2, lecturer);
//    test6 -> false (expected)
    if (!test6)
//        printCourses() is called to see if course2 is not added
        school.printCourses();

//    Expected Result: The addCourse method cannot function and
//    will return false because the code of course2 is the same
//    as the course code added before. (It would be like this even
//    if the name was the same.)
//    ##################################################################
```

Case 7: Removing the Course

```
//    ##################################################################
//    Case 7: Removing the Course
//    ------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);

    school.hireLecturer(lecturer);
    school.addCourse(course, lecturer);

    bool test7 = school.removeCourse(course);
//    test7 -> true (expected)
    if (test7)
//        printCourses() is called to see if Course is removed
        school.printCourses();

//    Expected Result: The removeCourse method performs its function
//    and removes the course from School. It also returns true.
//    ##################################################################
```

Case 8: Removing the Course (non-existing)

```
//    ##################################################################
//    Case 8: Removing the Course (non-existing)
```

```
//    ------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);

    bool test8 = school.removeCourse(course);
//    test8 -> false (expected)
    if (!test8)
//        printCourses() is called to see nothing
        school.printCourses();

//    Expected Result: The removeCourse method cannot perform its
//    function because the course is not opened by the school and
//    returns false.
//    ##################################################################
```

Case 9: Removing the Course by the Course Code

```
//    ##################################################################
//    Case 9: Removing the Course by the Course Code
//    ------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);

    school.hireLecturer(lecturer);
    school.addCourse(course, lecturer);

    bool test9 = school.removeCourseByCode("IBIM");
//    test9 -> true (expected)
    if (test9)
//        printCourses() is called to see nothing
        school.printCourses();

//    Expected Result: The removeCourseByCode method performs its
//    function (searches the added courses for the entered code
//    and removes it if found) and returns true.
//    ##################################################################
```

Case 10: Removing the Course by the Course Code (non-existing)

```
//    ##############################################################
//    Case 10: Removing the Course by the Course Code (non-existing)
//    --------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);

    bool test10 = school.removeCourseByCode("ECULT");
//    test10 -> false (expected)
    if (!test10)
//        printCourses() is called to see nothing
        school.printCourses();

//    Expected Result: The removeCourseByCode method cannot perform
//    its function because it cannot find the Add Course
//    corresponding to the entered code and returns false.
//    ##############################################################
```

Case 11: Registering a New Student

```
//    ##############################################################
//    Case 11: Registering a New Student
//    --------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");

//    create new Student object
    Student student("Peter Parker", "040237508");

    bool test11 =school.registerStudent(student);
//    test11 -> true (expected)
    if (test11)
//        printStudents() is called to see if student is registered
        school.printStudents();

//    Expected Result: The registerStudent method performs its
//    function and registers the student. returns true.
//    ##############################################################
```

Case 12: Registering a New Student (with same student number)

```
//    ##############################################################
//    Case 12: Registering a New Student (with same student number)
//    --------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");

//    create new Student object
    Student student("Peter Parker", "040237508");
    Student student2("Jack Sparrow", "040237508");

    school.registerStudent(student);

    bool test12 =school.registerStudent(student2);
//    test12 -> false (expected)
    if (!test12)
//        printStudents() is called to see if student2 is not
//        registered
        school.printStudents();

//    Expected Result: The registerStudent method cannot fulfill its
//    function because another student is registered to the school
//    with the same student number. and returns false. That's why
//    only student is printed on the screen.
//    ##############################################################
```

Case 13: Unregistering the Student

```
//    ##############################################################
//    Case 13: Unregistering the Student
//    --------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");

//    create new Student object
    Student student("Peter Parker", "040237508");

    school.registerStudent(student);

    bool test13 =school.unregisterStudent(student);
//    test13 -> true (expected)
    if (test13)
//        printStudents() is called to see nothing
        school.printStudents();
```

```
//    Expected Result: The unregisterStudent method performs its
//    function and unregisters the student from the school,
//    returning true. As a result, no students are printed on the
//    screen.
//    ###################################################################
```

Case 14: Unregistering the Student (non-existing)

```
//    ###################################################################
//    Case 14: Unregister the Student (non-existing)
//    -------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");

//    create new Student object
    Student student("Peter Parker", "040237508");

    bool test14 =school.unregisterStudent(student);
//    test14 -> false (expected)
    if (!test14)
//        printStudents() is called to see nothing
        school.printStudents();

//    Expected Result: The unregisterStudent method cannot perform
//    its function because the student is not registered at the
//    school and returns false. As a result, no students are printed
//    on the screen.
//    ###################################################################
```

Case 15: Unregistering the Student by the Student Number

```
//    ###################################################################
//    Case 15: Unregistering the Student by the Student Number
//    -------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");

//    create new Student object
    Student student("Peter Parker", "040237508");

    school.registerStudent(student);

    bool test15 =
school.unregisterStudentByStudentNumber("040237508");
//    test15 -> true (expected)
```

```
    if (test15)
//        printStudents() is called to see nothing
        school.printStudents();

//    Expected Result: The unregisterStudentByStudentNumber method
//    performs its function by finding the student corresponding
//    to the entered student number and deleting his/her
//    registration. Returns true. Additionally, since the student
//    record is deleted, no student will be displayed on the screen.
//    ###################################################################
```

Case 16: Unregistering the Student by the Student Number (non-existing)

```
//    ###################################################################
//    Case 16: Unregistering the Student by the Student Number
//    (non-existing)
//    -------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");

//    create new Student object
    Student student("Peter Parker", "040237508");

    school.registerStudent(student);

    bool test16 =
school.unregisterStudentByStudentNumber("040200871");
//    test16 -> false (expected)
    if (!test16)
//        printStudents() is called to see student
        school.printStudents();

//    Expected Result: The unregisterStudentByStudentNumber method
//    cannot perform its function because it cannot find a
//    registered student corresponding to the entered number.
//    As a result, it returns false and prints the previously
//    registered student on the screen.
//    ###################################################################
```

Case 17: Updating the Lecturer of the Course

```
//    ###################################################################
//    Case 17: Updating the Lecturer of the Course
//    -------------------------------------------------------------------
//    create new School object
```

```
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
    Lecturer lecturer2("Tolga Ozuygur", "Research Assistant",
                       "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);

    school.hireLecturer(lecturer);
    school.hireLecturer(lecturer2);
    school.addCourse(course, lecturer);

    bool test17 = course.updateLecturer(lecturer2);
//    test17 -> true (expected)
    if (test17)
//        getNameOfTheCourseLecturer() returns name of the Lecturer
        cout << course.getNameOfTheCourseLecturer();

//    Expected Result: The updateLecturer method performs its
//    function and changes the Lecturer of the course. It returns
//    true and prints the Course Lecturer name on the screen.
//    ###############################################################
```

Case 18: Updating the Lecturer of the Course (there is no currently course lecturer)

```
//    ###############################################################
//    Case 18: Updating the Lecturer of the Course (there is no
//    currently course lecturer)
//    ---------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
    Lecturer lecturer2("Tolga Ozuygur", "Research Assistant",
                       "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);

    school.hireLecturer(lecturer);
    school.hireLecturer(lecturer2);
    school.addCourse(course, lecturer);
    course.fireLecturerFromCourse();

    bool test18 = course.updateLecturer(lecturer2);
//    test18 -> false (expected)
```

```
    if (!test18)
//        getNameOfTheCourseLecturer() returns empty string
        cout << course.getNameOfTheCourseLecturer();

//    Expected Result: The updateLecturer method cannot function
//    because updating is only possible if there is an existing
//    Lecturer. The value false is returned and no name is written
//    to the screen.
//    ###################################################################
```

Case 19: Updating the Lecturer of the Course (maximum course capacity)

```
//    ###################################################################
//    Case 19: Updating the Lecturer of the Course (maximum course
//    capacity)
//    -------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
    Lecturer lecturer2("Tolga Ozuygur", "Research Assistant",
                       "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);
    Course course2("Advanced Topics in Building Iron Man", "ABIM",
4);
    Course course3("Introduction to Arc Reactor", "IAC", 3);
    Course course4("Introduction to Propulsion", "IPRO", 3);

    school.hireLecturer(lecturer);
    school.hireLecturer(lecturer2);
    school.addCourse(course, lecturer2);
    school.addCourse(course2, lecturer);
    school.addCourse(course3, lecturer);
    school.addCourse(course4, lecturer);

    bool test19 = course.updateLecturer(lecturer);
//    test19 -> false (expected)
    if (!test19)
//        getNameOfTheCourseLecturer() returns name of the lecturer
//        ("Tolga Ozuygur")
        cout << course.getNameOfTheCourseLecturer();

//    Expected Result: The updateLecturer method cannot fulfill its
//    function because lecturer2 has reached its capacity by giving
```

```
//    3 lectures in total and cannot give the 4th lecture. The
//    value false is returned and "Tolga Ozuygur" is printed on
//    the screen.
//    ##################################################################
```

Case 20: Updating the Lecturer of the Course (the lecturer is not employed by school)

```
//    ##################################################################
//    Case 20: Updating the Lecturer of the Course (the lecturer is
//    not employed by school)
//    ------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
    Lecturer lecturer2("Tolga Ozuygur", "Research Assistant",
                       "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);

    school.hireLecturer(lecturer);
    school.addCourse(course, lecturer);

    bool test20 = course.updateLecturer(lecturer2);
//    test20 -> false (expected)
    if (!test20)
//        getNameOfTheCourseLecturer() returns empty string
        cout << course.getNameOfTheCourseLecturer();

//    Expected Result: The updateLecturer method cannot perform its
//    function because the lecturer2 does not work at the school.
//    The value false is returned and "Tony Stark" is printed
//    on the screen.
//    ##################################################################
```

Case 21: Updating the Lecturer of the Course (course is not active)

```
//    ##################################################################
//    Case 21: Updating the Lecturer of the Course
//    (course is not active)
//    ------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
```

```
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
    Lecturer lecturer2("Tolga Ozuygur", "Research Assistant",
                       "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);

    school.hireLecturer(lecturer);
    school.hireLecturer(lecturer2);

    bool test21 = course.updateLecturer(lecturer2);
//    test21 -> false (expected)
    if (!test21)
//        getNameOfTheCourseLecturer() returns empty string
        cout << course.getNameOfTheCourseLecturer();

//    Expected Result: The updateLecturer function cannot perform
//    its function because the course has not been added by the
//    school, that is, it is not an active course, and as a result,
//    it returns false. Nothing is printed on the screen.
//    ###################################################################
```

Case 22: Firing the Lecturer from the Course

```
//    ###################################################################
//    Case 22: Firing the Lecturer from the Course
//    -------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);

    school.hireLecturer(lecturer);
    school.addCourse(course, lecturer);

    bool test22 = course.fireLecturerFromCourse();
//    test22 -> true (expected)
    if (test22)
//        getNameOfTheCourseLecturer() returns empty string
        cout << course.getNameOfTheCourseLecturer();

//    Expected Result: fireLecturerFromCourse method executes and
//    terminates the lecturer's course task, returning true. Nothing
//    is printed on the screen because there is no lecturer for the
```

```
//    course anymore.
//    ###################################################################
```

Case 23: Firing the Lecturer from the Course (there is no currently course lecturer)

```
//    ###################################################################
//    Case 23: Firing the Lecturer from the Course
//    (there is no currently course lecturer)
//    -------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);

    school.hireLecturer(lecturer);
    school.addCourse(course, lecturer);
    course.fireLecturerFromCourse();

    bool test23 = course.fireLecturerFromCourse();
//    test23 -> false (expected)
    if (!test23)
//        getNameOfTheCourseLecturer() returns empty string
        cout << course.getNameOfTheCourseLecturer();

//    Expected Result: fireLecturerFromCourse cannot perform the
//    function because the course does not currently have a
//    lecturer. It returns false and nothing is printed to the
//    screen.
//    ###################################################################
```

Case 24: Firing the Lecturer from the Course (course is not active)

```
//    ###################################################################
//    Case 24: Firing the Lecturer from the Course
//    (course is not active)
//    -------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
```

```
    Course course("Introduction to Building Iron Man", "IBIM", 4);

    school.hireLecturer(lecturer);
    school.addCourse(course, lecturer);
    school.removeCourse(course);

    bool test24 = course.fireLecturerFromCourse();
//    test24 -> false (expected)
    if (!test24)
//        getNameOfTheCourseLecturer() returns empty string
        cout << course.getNameOfTheCourseLecturer();

//    Expected Result: fireLecturerFromCourse cannot perform its
//    function because the course is not currently offered by
//    the school. It returns false and does not print anything
//    to the screen.
//    ##################################################################
```

Case 25: Assigning a Lecturer to the Course

```
//    ##################################################################
//    Case 25: Assigning a Lecturer to the Course
//    (course is not active)
//    ------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
    Lecturer lecturer2("Tolga Ozuygur", "Research Assistant",
                       "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);

    school.hireLecturer(lecturer);
    school.hireLecturer(lecturer2);
    school.addCourse(course, lecturer);
    course.fireLecturerFromCourse();

    bool test25 = course.assignLecturer(lecturer2);
//    test25 -> true (expected)
    if (test25)
//        getNameOfTheCourseLecturer() returns name of the lecturer2
        cout << course.getNameOfTheCourseLecturer();

//    Expected Result: The assignLecturer method performs its
//    function and assigns a lecturer to a course offered by the
```

```
//    school that does not have a lecturer. Returns true and prints
//    the name of the newly assigned Lecturer.
//    ###################################################################
```

Case 26: Assigning a Lecturer to the Course (the lecturer is not employed by school)

```
//    ###################################################################
//    Case 26: Assigning a Lecturer to the Course
//    (the lecturer is not employed by school)
//    -------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
    Lecturer lecturer2("Tolga Ozuygur", "Research Assistant",
                       "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);

    school.hireLecturer(lecturer);
    school.addCourse(course, lecturer);
    course.fireLecturerFromCourse();

    bool test26 = course.assignLecturer(lecturer2);
//    test26 -> false (expected)
    if (!test26)
//        getNameOfTheCourseLecturer() returns empty string
        cout << course.getNameOfTheCourseLecturer();

//    Expected Result: The assignLecturer method cannot fulfill its
//    function because the lecturer to be assigned does not work at
//    the school. It returns false and does not print anything to
//    the screen.
//    ###################################################################
```

Case 27: Assigning a Lecturer to the Course (maximum course capacity)

```
//    ###################################################################
//    Case 27: Assigning a Lecturer to the Course
//    (maximum course capacity)
//    -------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
```

```
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
    Lecturer lecturer2("Tolga Ozuygur", "Research Assistant",
                       "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);
    Course course2("Advanced Topics in Building Iron Man", "ABIM",
4);
    Course course3("Introduction to Arc Reactor", "IAC", 3);
    Course course4("Introduction to Propulsion", "IPRO", 3);

    school.hireLecturer(lecturer);
    school.hireLecturer(lecturer2);
    school.addCourse(course, lecturer2);
    school.addCourse(course2, lecturer);
    school.addCourse(course3, lecturer);
    school.addCourse(course4, lecturer);

    bool test27 = course.assignLecturer(lecturer2);
//    test27 -> false (expected)
    if (!test27)
//        getNameOfTheCourseLecturer() returns empty string
        cout << course.getNameOfTheCourseLecturer();

//    Expected Result: The assignLecturer method cannot fulfill
//    its function because the lecturer to be assigned gives
//    the maximum number of lectures. It returns false and does
//    not print anything to
//    the screen.
//    ###############################################################
```

Case 28: Assigning a Lecturer to the Course (course is not active)

```
//    ###############################################################
//    Case 28: Assigning a Lecturer to the Course
//    (course is not active)
//    ---------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
    Lecturer lecturer2("Tolga Ozuygur", "Research Assistant",
                       "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);
```

```
    school.hireLecturer(lecturer);
    school.hireLecturer(lecturer2);
    school.addCourse(course, lecturer);
    school.removeCourse(course);

    bool test28 = course.assignLecturer(lecturer2);
//    test28 -> false (expected)
    if (!test28)
//        getNameOfTheCourseLecturer() returns empty string
        cout << course.getNameOfTheCourseLecturer();

//    Expected Result: The assignLecturer method cannot function
//    because the course is not offered by the school, meaning it is
//    not active. It returns false and does not print anything to
//    the screen.
//    ###################################################################
```

Case 29: Quitting Teaching the Course

```
//    ###################################################################
//    Case 29: Quitting Teaching the Course
//    -------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);

    school.hireLecturer(lecturer);
    school.addCourse(course, lecturer);

    bool test29 = lecturer.quitTeachingTheCourse(course);
//    test29 -> true (expected)
    if (test29)
//        printCourses print course(s) given by lecturer
        lecturer.printCourses();

//    Expected Result: The quitTeachingTheCourse method executes its
//    function and terminates the Lecturer's task in the course. It
//    functions the same as the fireLecturerFromCourse function of
//    the Course class. It returns true and does not print anything
//    to the screen.
//    ###################################################################
```

Case 30: Quitting Teaching the Course (the lecturer is not employed by school)

```
//    ##################################################################
//    Case 30: Quitting Teaching the Course
//    (the lecturer is not employed by school)
//    ------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);

    school.hireLecturer(lecturer);
    school.addCourse(course, lecturer);
    school.fireLecturer(lecturer);

    bool test30 = lecturer.quitTeachingTheCourse(course);
//    test30 -> false (expected)
    if (!test30)
//        printCourses print course(s) given by lecturer
        lecturer.printCourses();

//    Expected Result: The quitTeachingTheCourse method cannot
//    perform its function because the lecturer is not actively
//    working at the school. It returns false and does not print
//    anything to the screen because, in this scenario, the course
//    instructor has left the school.
//    ##################################################################
```

Case 31: Quitting Teaching the Course (the lecturer of the course is not (this) lecturer)

```
//    ##################################################################
//    Case 31: Quitting Teaching the Course
//    (the lecturer of the course is not (this) lecturer)
//    ------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Lecturer lecturer2("Tolga Ozuygur", "Research Assistant",
                       "Electronics");
```

```
    Course course("Introduction to Building Iron Man", "IBIM", 4);

    school.hireLecturer(lecturer);
    school.hireLecturer(lecturer2);
    school.addCourse(course, lecturer);

    bool test31 = lecturer2.quitTeachingTheCourse(course);
//    test31 -> false (expected)
    if (!test31)
//        printCourses print course(s) given by lecturer
        lecturer.printCourses();

//    Expected Result: The quitTeachingTheCourse method cannot
//    function because lecturer2 is not the lecturer of the course.
//    It returns false and prints the course
//    ("Introduction to Building Iron Man") given by the lecturer
//    ("Tony Stark") on the screen.
//    ##################################################################
```

Case 32: Quitting Teaching the Course (course is not active)

```
//    ##################################################################
//    Case 32: Quitting Teaching the Course
//    (course is not active)
//    ------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);

    school.hireLecturer(lecturer);
    school.addCourse(course, lecturer);
    school.removeCourse(course);

    bool test32 = lecturer.quitTeachingTheCourse(course);
//    test32 -> false (expected)
    if (!test32)
//        printCourses print course(s) given by lecturer
        lecturer.printCourses();

//    Expected Result: The quitTeachingTheCourse method cannot
//    fulfill its function because the course is no longer offered
//    by the school, so the lecturer cannot be the lecturer of the
//    inactive course. It returns false and in this scenario,
```

```
//    nothing is printed because the lecturer does not offer another
//    course.
//    ###################################################################
```

Case 33: Becoming Lecturer of the Course

```
//    ###################################################################
//    Case 33: Becoming Lecturer of the Course
//    ------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
    Lecturer lecturer2("Tolga Ozuygur", "Research Assistant",
                       "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);

    school.hireLecturer(lecturer);
    school.hireLecturer(lecturer2);
    school.addCourse(course, lecturer);
    lecturer.quitTeachingTheCourse(course);

    bool test33 = lecturer2.becomeLecturerOfCourse(course);
//    test33 -> true (expected)
    if (test33)
//        printCourses print course(s) given by lecturer2
        lecturer2.printCourses();

//    Expected Result: The becomeLecturerOfCourse method performs
//    its function and lecturer2 becomes the lecturer of the course
//    that does not have a lecturer. It returns true and this course
//    started by lecturer2 is printed on the screen.
//    ###################################################################
```

Case 34: Becoming Lecturer of the Course (the lecturer is not employed by school)

```
//    ###################################################################
//    Case 34: Becoming Lecturer of the Course
//    (the lecturer is not employed by school)
//    ------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
```

```
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
    Lecturer lecturer2("Tolga Ozuygur", "Research Assistant",
                       "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);

    school.hireLecturer(lecturer);
    school.addCourse(course, lecturer);
    lecturer.quitTeachingTheCourse(course);

    bool test34 = lecturer2.becomeLecturerOfCourse(course);
//    test34 -> false (expected)
    if (!test34)
//        printCourses print course(s) given by lecturer2
        lecturer2.printCourses();

//    Expected Result: The becomeLecturerOfCourse method is executed
//    and lecturer2 is not currently employed at the school (he may
//    have never been hired or may have been fired). It returns
//    false and in this scenario nothing is printed to the screen
//    because lecturer2 cannot work at the school.
//    ###############################################################
```

Case 35: Becoming Lecturer of the Course (maximum course capacity)

```
//    ###############################################################
//    Case 35: Becoming Lecturer of the Course
//    (maximum course capacity)
//    ---------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
    Lecturer lecturer2("Tolga Ozuygur", "Research Assistant",
                       "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);
    Course course2("Advanced Topics in Building Iron Man", "ABIM",
4);
    Course course3("Introduction to Arc Reactor", "IAC", 3);
    Course course4("Introduction to Propulsion", "IPRO", 3);

    school.hireLecturer(lecturer);
    school.hireLecturer(lecturer2);
    school.addCourse(course, lecturer);
```

```
    school.addCourse(course2, lecturer);
    school.addCourse(course3, lecturer);
    school.addCourse(course4, lecturer2);
    lecturer2.quitTeachingTheCourse(course4);

    bool test35 = lecturer.becomeLecturerOfCourse(course4);
//    test35 -> false (expected)
    if (!test35)
//        printCourses print course(s) given by lecturer
        lecturer.printCourses();

//    Expected Result: The becomeLecturerOfCourse method cannot
//    function because the lecturer has reached the maximum number
//    of courses he can teach. It returns false and course, course2
//    and course3 are printed to the screen.
//    ################################################################
```

Case 36: Becoming Lecturer of the Course (the course may already have a lecturer)

```
//    ################################################################
//    Case 36: Becoming Lecturer of the Course
//    (the course may already have a lecturer)
//    ----------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
    Lecturer lecturer2("Tolga Ozuygur", "Research Assistant",
                       "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);

    school.hireLecturer(lecturer);
    school.hireLecturer(lecturer2);
    school.addCourse(course, lecturer);

    bool test36 = lecturer2.becomeLecturerOfCourse(course);
//    test36 -> false (expected)
    if (!test36)
//        printCourses print course(s) given by lecturer2
        lecturer2.printCourses();

//    Expected Result: The becomeLecturerOfCourse method cannot
//    perform because the course already has a lecturer.
//    It returns false and nothing is printed to the screen in
```

```
//    this scenario.
//    ##################################################################
```

Case 37: Becoming Lecturer of the Course (course is not active)

```
//    ##################################################################
//    Case 37: Becoming Lecturer of the Course
//    (course is not active)
//    ------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
    Lecturer lecturer2("Tolga Ozuygur", "Research Assistant",
                       "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);

    school.hireLecturer(lecturer);
    school.hireLecturer(lecturer2);
    school.addCourse(course, lecturer);
    school.removeCourse(course);

    bool test37 = lecturer2.becomeLecturerOfCourse(course);
//    test37 -> false (expected)
    if (!test37)
//        printCourses print course(s) given by lecturer2
        lecturer2.printCourses();

//    Expected Result: The becomeLecturerOfCourse method cannot
//    function because the course is no longer offered by the
//    school (it may never have been offered). It returns false and
//    nothing is printed to the screen in this scenario.
//    ##################################################################
```

Case 38: Adding a Student to the Course by the School

```
//    ##################################################################
//    Case 38: Adding a Student to the Course by the School
//    ------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
```

```
    Course course("Introduction to Building Iron Man", "IBIM", 4);
//    create new Student object
    Student student("Peter Parker", "040237508");

    school.hireLecturer(lecturer);
    school.registerStudent(student);
    school.addCourse(course, lecturer);

    bool test38 = school.addStudentToCourse(student, course);
//    test38 -> true (expected)
    if (test38)
//        printCourses print course(s) taken by the student
        student.printCourses();

//    Expected Result: The addStudentToCourse method enrolls the
//    student in the course. It establishes the connections between
//    the students and the student becomes a student of that course.
//    Returns true and the course taken by the student is printed on
//    the screen.
//    ################################################################
```

Case 39: Adding a Student to the Course by the School (the student is not registered)

```
//    ################################################################
//    Case 39: Adding a Student to the Course by the School
//    (the student is not registered)
//    ----------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);
//    create new Student object
    Student student("Peter Parker", "040237508");

    school.hireLecturer(lecturer);
    school.addCourse(course, lecturer);

    bool test39 = school.addStudentToCourse(student, course);
//    test39 -> false (expected)
    if (!test39)
//        printCourses print course(s) taken by the student
        student.printCourses();
```

```
//    Expected Result: addStudentToCourse cannot perform its
//    function because the student to be enrolled in the course
//    is not registered at the school. It returns false and the
//    course cannot be printed to the screen in this scenario.
//    ##################################################################
```

Case 40: Adding a Student to the Course by the School (the course at full capacity)

```
//    ##################################################################
//    Case 40: Adding a Student to the Course by the School
//    (the course at full capacity)
//    ------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);
//    create new Student object
    Student student("Peter Parker", "040237550");
    Student student1("Emily Johnson", "040237501");
    Student student2("Michael Brown", "040237502");
    Student student3("Emma Williams", "040237503");
    Student student4("Daniel Jones", "040237504");
    Student student5("Sophia Davis", "040237505");
    Student student6("Matthew Wilson", "040237506");
    Student student7("Olivia Taylor", "040237507");
    Student student8("James Martinez", "040237508");
    Student student9("Isabella Lopez", "040237509");
    Student student10("William Anderson", "040237510");

    school.hireLecturer(lecturer);
    school.registerStudent(student);
    school.registerStudent(student1);
    school.registerStudent(student2);
    school.registerStudent(student3);
    school.registerStudent(student4);
    school.registerStudent(student5);
    school.registerStudent(student6);
    school.registerStudent(student7);
    school.registerStudent(student8);
    school.registerStudent(student9);
    school.registerStudent(student10);
    school.addCourse(course, lecturer);
```

```
school.addStudentToCourse(student1, course);
school.addStudentToCourse(student2, course);
school.addStudentToCourse(student3, course);
school.addStudentToCourse(student4, course);
school.addStudentToCourse(student5, course);
school.addStudentToCourse(student6, course);
school.addStudentToCourse(student7, course);
school.addStudentToCourse(student8, course);
school.addStudentToCourse(student9, course);
school.addStudentToCourse(student10, course);

    bool test40 = school.addStudentToCourse(student, course);
//    test40 -> false (expected)
    if (!test40)
//        printCourses print course(s) taken by the student
        student.printCourses();

//    Expected Result: addStudentToCourse cannot function because
//    10 students are currently enrolled in the course, indicating
//    that the course has reached maximum capacity. As a result,
//    student("Peter Parker") cannot register for the course. It
//    returns false and the course cannot be printed to the screen
//    in this scenario.
//    ################################################################
```

Case 41: Adding a Student to the Course by the School (the student is already enrolled the course)

```
//    ################################################################
//    Case 41: Adding a Student to the Course by the School
//    (the student is already enrolled the course)
//    ----------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);
//    create new Student object
    Student student("Peter Parker", "040237508");

    school.hireLecturer(lecturer);
    school.registerStudent(student);
    school.addCourse(course, lecturer);
    school.addStudentToCourse(student, course);
```

```
    bool test41 = school.addStudentToCourse(student, course);
//    test41 -> false (expected)
    if (!test41)
//        printCourses print course(s) taken by the student
        student.printCourses();

//    Expected Result: addStudentToCourse cannot perform its
//    function because the student to be enrolled in the course
//    is already registered in the course. It returns false and
//    in this scenario, the course taken by the student is printed
//    on the screen.
//    ###################################################################
```

Case 42: Adding a Student to the Course by the School (course is not active)

```
//    ###################################################################
//    Case 42: Adding a Student to the Course by the School
//    (course is not active))
//    -------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);
//    create new Student object
    Student student("Peter Parker", "040237508");

    school.hireLecturer(lecturer);
    school.registerStudent(student);
    school.addCourse(course, lecturer);
    school.removeCourse(course);

    bool test42 = school.addStudentToCourse(student, course);
//    test42 -> false (expected)
    if (!test42)
//        printCourses print course(s) taken by the student
        student.printCourses();

//    Expected Result: addStudentToCourse cannot perform its
//    function because the relevant course may not have been
//    opened by the school, may have been closed, or may not
//    have a Lecturer. It returns false and nothing is printed to
//    this screen.
//    ###################################################################
```

Case 43: Removing the Student from the Course by the School

```
//    ##################################################################
//    Case 43: Removing the Student from the Course by the School
//    ------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);
//    create new Student object
    Student student("Peter Parker", "040237508");

    school.hireLecturer(lecturer);
    school.registerStudent(student);
    school.addCourse(course, lecturer);
    school.addStudentToCourse(student, course);

    bool test43 = school.removeStudentFromCourse(student, course);
//    test43 -> true (expected)
    if (test43)
//        printCourses print course(s) taken by the student
        student.printCourses();

//    Expected Result: The removeStudentFromCourse method performs
//    its function and deletes the student who has previously
//    registered for the course from the course. It returns true
//    and no courses are printed because the student is no longer
//    enrolled in any courses in this scenario.
//    ##################################################################
```

Case 44: Removing the Student from the Course by the School (the student is not registered)

```
//    ##################################################################
//    Case 44: Removing the Student from the Course by the School
//    (the student is not registered)
//    ------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
```

```
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);
//    create new Student object
    Student student("Peter Parker", "040237508");

    school.hireLecturer(lecturer);
    school.registerStudent(student);
    school.addCourse(course, lecturer);
    school.addStudentToCourse(student, course);
    school.unregisterStudent(student);

    bool test44 = school.removeStudentFromCourse(student, course);
//    test44 -> false (expected)
    if (!test44)
//        printCourses print course(s) taken by the student
        student.printCourses();

//    Expected Result: It cannot perform the removeStudentFromCourse
//    function because the student whose registration is requested
//    to be deleted from the course is not currently registered at
//    the school, or may have never been registered. It returns
//    False and no courses are printed to the screen.
//    ###################################################################
```

Case 45: Removing the Student from the Course by the School (the student is already unenrolled the course)

```
//    ###################################################################
//    Case 45: Removing the Student from the Course by the School
//    (the student is already unEnrolled the course)
//    -------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);
//    create new Student object
    Student student("Peter Parker", "040237508");

    school.hireLecturer(lecturer);
    school.registerStudent(student);
    school.addCourse(course, lecturer);
    school.addStudentToCourse(student, course);
    school.removeStudentFromCourse(student, course);
```

```
    bool test45 = school.removeStudentFromCourse(student, course);
//    test45 -> false (expected)
    if (!test45)
//        printCourses print course(s) taken by the student
        student.printCourses();

//    Expected Result: It cannot perform the removeStudentFromCourse
//    function because the student whose enrollment is requested to
//    be deleted from the course has already been enEnrolled from
//    the course or may not have been enrolled in the course at all.
//    It returns False and no courses are printed to the screen.
//    ################################################################
```

Case 46: Removing the Student from the Course by the School (the course is not active)

```
//    ################################################################
//    Case 46: Removing the Student from the Course by the School
//    (the course is not active)
//    ----------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);
//    create new Student object
    Student student("Peter Parker", "040237508");

    school.hireLecturer(lecturer);
    school.registerStudent(student);
    school.addCourse(course, lecturer);
    school.addStudentToCourse(student, course);
    school.removeCourse(course);

    bool test46 = school.removeStudentFromCourse(student, course);
//    test46 -> false (expected)
    if (!test46)
//        printCourses print course(s) taken by the student
        student.printCourses();

//    Expected Result: t cannot perform the removeStudentFromCourse
//    function because the course for which the student's
//    enrollment is requested has been closed by the school or
//    may not have been opened at all. It returns false and no
```

```
//    courses are written to the screen in this scenario.
//    ###################################################################
```

Case 47: Adding a Student to the Course by the Course

```
//    ###################################################################
//    Case 47: Adding a Student to the Course by the Course
//    -------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);
//    create new Student object
    Student student("Peter Parker", "040237508");

    school.hireLecturer(lecturer);
    school.registerStudent(student);
    school.addCourse(course, lecturer);

    bool test47 = course.addStudent(student);
//    test47 -> true (expected)
    if (test47)
//        printCourses print course(s) taken by the student
        student.printCourses();

//    Expected Result: The addStudent method performs its function
//    and the student is added to the course. It returns true and
//    the course taken by the student is printed on the screen.
//    ###################################################################
```

Case 48: Adding a Student to the Course by the Course (the student is not registered)

```
//    ###################################################################
//    Case 48: Adding a Student to the Course by the Course
//    (the student is not registered)
//    -------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
```

```
    Course course("Introduction to Building Iron Man", "IBIM", 4);
//    create new Student object
    Student student("Peter Parker", "040237508");

    school.hireLecturer(lecturer);
    school.addCourse(course, lecturer);

    bool test48 = course.addStudent(student);
//    test48 -> false (expected)
    if (!test48)
//        printCourses print course(s) taken by the student
        student.printCourses();

//    Expected Result: The addStudent method cannot fulfill its
//    function because the student to be added to the course is
//    not registered to the school. It returns false and nothing
//    is printed to the screen.
//    ###################################################################
```

Case 49: Adding a Student to the Course by the Course (the course at full capacity)

```
//    ###################################################################
//    Case 49: Adding a Student to the Course by the Course (the
course at full capacity)
//    -------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);
//    create new Student object
    Student student("Peter Parker", "040237550");
    Student student1("Emily Johnson", "040237501");
    Student student2("Michael Brown", "040237502");
    Student student3("Emma Williams", "040237503");
    Student student4("Daniel Jones", "040237504");
    Student student5("Sophia Davis", "040237505");
    Student student6("Matthew Wilson", "040237506");
    Student student7("Olivia Taylor", "040237507");
    Student student8("James Martinez", "040237508");
    Student student9("Isabella Lopez", "040237509");
    Student student10("William Anderson", "040237510");

    school.hireLecturer(lecturer);
```

```
    school.registerStudent(student);
    school.registerStudent(student1);
    school.registerStudent(student2);
    school.registerStudent(student3);
    school.registerStudent(student4);
    school.registerStudent(student5);
    school.registerStudent(student6);
    school.registerStudent(student7);
    school.registerStudent(student8);
    school.registerStudent(student9);
    school.registerStudent(student10);
    school.addCourse(course, lecturer);
    course.addStudent(student1);
    course.addStudent(student2);
    course.addStudent(student3);
    course.addStudent(student4);
    course.addStudent(student5);
    course.addStudent(student6);
    course.addStudent(student7);
    course.addStudent(student8);
    course.addStudent(student9);
    course.addStudent(student10);

    bool test49 = course.addStudent(student);
//    test49 -> false (expected)
    if (!test49)
//        printCourses print course(s) taken by the student
        student.printCourses();

//    Expected Result: The addStudent method cannot function
//    because the course has reached maximum capacity and a new
//    student cannot enroll in the course. It returns false and
//    nothing is printed to the screen.
//    ################################################################
```

Case 50: Adding a Student to the Course by the Course (the student is already enrolled the course)

```
//    ################################################################
//    Case 50: Adding a Student to the Course by the Course
//    (the student is already enrolled the course)
//    ----------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
```

```
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);
//    create new Student object
    Student student("Peter Parker", "040237508");

    school.hireLecturer(lecturer);
    school.registerStudent(student);
    school.addCourse(course, lecturer);
    course.addStudent(student);

    bool test50 = course.addStudent(student);
//    test50 -> false (expected)
    if (!test50)
//        printCourses print course(s) taken by the student
        student.printCourses();

//    Expected Result: The addStudent method cannot perform its
//    function because the student to be registered is already
//    registered in the course. It returns false and the course
//    it has already taken is printed on the screen.
//    ################################################################
```

Case 51: Adding a Student to the Course by the Course (course is not active)

```
//    ################################################################
//    Case 51: Adding a Student to the Course by the Course
//    (course is not active)
//    ----------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);
//    create new Student object
    Student student("Peter Parker", "040237508");

    school.hireLecturer(lecturer);
    school.registerStudent(student);
    school.addCourse(course, lecturer);
    school.removeCourse(course);

    bool test51 = course.addStudent(student);
//    test51 -> false (expected)
    if (!test51)
```

```
//        printCourses print course(s) taken by the student
        student.printCourses();

//    Expected Result: The addStudent method cannot function
//    because the course has been closed by the school (maybe
//    it was never opened). It returns false and nothing is printed
//    to the screen.
//    ###################################################################
```

Case 52: Removing the Student from the Course by the Course

```
//    ###################################################################
//    Case 52: Removing the Student from the Course by the Course
//    -------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);
//    create new Student object
    Student student("Peter Parker", "040237508");

    school.hireLecturer(lecturer);
    school.registerStudent(student);
    school.addCourse(course, lecturer);
    course.addStudent(student);

    bool test52 = course.removeStudent(student);
//    test52 -> true (expected)
    if (test52)
//        printCourses print course(s) taken by the student
        student.printCourses();

//    Expected Result: The removeStudent method performs its
//    function and removes the student previously added to the
//    course from the course. It returns true and nothing is
//    printed because the student has been removed from the course.
//    ###################################################################
```

Case 53: Removing the Student from the Course by the Course (the student is not registered)

```
//    ##################################################################
//    Case 53: Removing the Student from the Course by the Course
//    (the student is not registered)
//    ------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);
//    create new Student object
    Student student("Peter Parker", "040237508");

    school.hireLecturer(lecturer);
    school.registerStudent(student);
    school.addCourse(course, lecturer);
    course.addStudent(student);
    school.unregisterStudent(student);

    bool test53 = course.removeStudent(student);
//    test53 -> false (expected)
    if (!test53)
//        printCourses print course(s) taken by the student
        student.printCourses();

//    Expected Result: The removeStudent method cannot function
//    because the student to be removed from the course is not
//    already enrolled in the school (or has been de-enrolled),
//    as a result, the student is no longer an active student.
//    It returns false and nothing is printed to the screen.
//    ##################################################################
```

Case 54: Removing the Student from the Course by the Course (the student is already unenrolled the course)

```
//    ##################################################################
//    Case 54: Removing the Student from the Course by the Course
//    (the student is already unEnrolled the course)
//    ------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
```

```
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);
//    create new Student object
    Student student("Peter Parker", "040237508");

    school.hireLecturer(lecturer);
    school.registerStudent(student);
    school.addCourse(course, lecturer);
    course.addStudent(student);
    course.removeStudent(student);

    bool test54 = course.removeStudent(student);
//    test54 -> false (expected)
    if (!test54)
//        printCourses print course(s) taken by the student
        student.printCourses();

//    Expected Result: The removeStudent method cannot fulfill its
//    function because the student to be removed from the course
//    has already left the course or has been removed (via methods
//    of other objects). It returns false and nothing is printed to
//    the screen.
//    ################################################################
```

Case 55: Removing the Student from the Course by the Course (the course is not active)

```
//    ################################################################
//    Case 55: Removing the Student from the Course by the Course
//    (the course is not active)
//    ----------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);
//    create new Student object
    Student student("Peter Parker", "040237508");

    school.hireLecturer(lecturer);
    school.registerStudent(student);
    school.addCourse(course, lecturer);
    course.addStudent(student);
    school.removeCourse(course);
```

```
    bool test55 = course.removeStudent(student);
//    test55 -> false (expected)
    if (!test55)
//        printCourses print course(s) taken by the student
        student.printCourses();

//    Expected Result: The removeStudent method cannot perform its
//    function because the course has been removed by the school,
//    that is, it is not active (it may not have been opened by the
//    school at all). It returns false and nothing is printed to
//    the screen.
//    ################################################################
```

Case 56: Changing Number of Credits the Course

```
//    ################################################################
//    Case 56: Changing Number of Credits the Course
//    ----------------------------------------------------------------
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);

    bool test56 = course.changeNumberOfCredits(3);
//    test56 -> true (expected)
    if (test56)
//        getNumberOfCredits() returns number of credits of the
//        (this) course
        cout << course.getNumberOfCredits();

//    Expected Result: The changeNumberOfCredits method performs
//    its function and updates the number of credits with the number
//    entered as an argument. Since the number is within the
//    appropriate range, the transaction has been completed
//    successfully. It returns true and the number of credits for
//    the course is printed on the screen.
//    ################################################################
```

Case 57: Changing Number of Credits the Course (invalid input)

```
//    ################################################################
//    Case 57: Changing Number of Credits the Course
//    (invalid input)
//    ----------------------------------------------------------------
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);
```

```
    bool test57 = course.changeNumberOfCredits(-3);
//    test57 -> false (expected)
    if (!test57)
//        getNumberOfCredits() returns number of credits of the
//        (this) course
        cout << course.getNumberOfCredits();

//    Expected Result: The changeNumberOfCredits method cannot
//    function because a suitable number of credits has not been
//    entered (-3), and it would not have been able to do so without
//    the appropriate range (it could be between 1 and 4). It
//    returns false and the old (unchanged) credit number of the
//    course is printed on the screen.
//    ################################################################
```

Case 58: Registering to the School by Student Object

```
//    ################################################################
//    Case 58: Registering to the School by Student Object
//    ----------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Student object
    Student student("Peter Parker", "040237508");

    bool test58 = student.registerToSchool(school);
//    test58 -> true (expected)
    if (test58)
//        printStudents print student(s) registered in school
        school.printStudents();

//    Expected Result: The registerToSchool method performs its
//    function and the student is registered to the school entered
//    as an argument. It functions the same as the registerStudent
//    method of the School class. The method returns true and the
//    registered student is printed on the screen.
//    ################################################################
```

Case 59: Registering to the School by Student Object (currently registered)

```
//    ################################################################
//    Case 59: Registering to the School by Student Object
```

```
//     (currently registered)
//     ----------------------------------------------------------------
//     create new School object
    School school("Warsaw University of Technology");
//     create new Student object
    Student student("Peter Parker", "040237508");

    student.registerToSchool(school);

    bool test59 = student.registerToSchool(school);
//     test59 -> false (expected)
    if (!test59)
//         printStudents print student(s) registered in school
        school.printStudents();

//     Expected Result: The registerToSchool method cannot perform
//     its function because the student is already enrolled in the
//     school. The method returns false and the student who is
//     already enrolled in the school is printed on the screen.
//     ################################################################
```

Case 60: Registering to the School by Student Object (there is a student with same student number)

```
//     ################################################################
//     Case 60: Registering to the School by Student Object
//     (there is a student with same student number)
//     ----------------------------------------------------------------
//     create new School object
    School school("Warsaw University of Technology");
//     create new Student object
    Student student("Peter Parker", "040237508");
    Student student2("Jack Sparrow", "040237508");

    student.registerToSchool(school);

    bool test60 = student2.registerToSchool(school);
//     test60 -> false (expected)
    if (!test60)
//         printStudents print student(s) registered in school
        school.printStudents();

//     Expected Result: The registerToSchool method cannot fulfill
//     its function because another student with the same number as
//     the student who wants to register to the school is registered
//     to the school. As a result, he cannot be enrolled in school.
```

```
//    The method returns false and the student who is already
//    enrolled in the school is printed on the screen.
//    ################################################################
```

Case 61: Unregistering from the School by Student Object

```
//    ################################################################
//    Case 61: Unregistering from the School by Student Object
//    ----------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Student object
    Student student("Peter Parker", "040237508");

    student.registerToSchool(school);

    bool test61 = student.unregisterFromSchool();
//    test61 -> true (expected)
    if (test61)
//        printStudents print student(s) registered in school
        school.printStudents();

//    Expected Result: The unregisterFromSchool method performs its
//    function and deletes the registered student's school record.
//    The function returns true because it fulfills its main task,
//    but no student is printed on the screen because the registered
//    student's record has been deleted.
//    ################################################################
```

Case 62: Unregistering from the School by Student Object (currently unregistered)

```
//    ################################################################
//    Case 62: Unregistering from the School by Student Object
//    (currently unregistered)
//    ----------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Student object
    Student student("Peter Parker", "040237508");

    student.registerToSchool(school);
    student.unregisterFromSchool();
```

```
    bool test62 = student.unregisterFromSchool();
//    test62 -> false (expected)
    if (!test62)
//        printStudents print student(s) registered in school
        school.printStudents();

//    Expected Result: The unregisterFromSchool method cannot
//    function because the student's registration has already been
//    deleted (or the student may not have been enrolled at all).
//    It returns false and no students are printed on the screen.
//    ################################################################
```

Case 63: Enrolling to the Course by Student Object

```
//    ################################################################
//    Case 63: Enrolling to the Course by Student Object
//    ----------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);
//    create new Student object
    Student student("Peter Parker", "040237508");

    school.hireLecturer(lecturer);
    school.registerStudent(student);
    school.addCourse(course, lecturer);

    bool test63 = student.enrollCourse(course);
//    test63 -> true (expected)
    if (test63)
//        printStudents() print student(s) enrolled in course.
        course.printStudents();

//    Expected Result: The enrollCourse method performs its function
//    and the student enrolls in the course (it performs the same
//    function as the addStudentToCourse() method of the School
//    class or the addStudent() method of the Course class.). It
//    returns true and the student registered for the course is
//    printed.
//    ################################################################
```

Case 64: Enrolling to the Course by Student Object (the student is not registered in school)

```
//    ##################################################################
//    Case 64: Enrolling to the Course by Student Object
//    (the student is not registered in school)
//    ------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);
//    create new Student object
    Student student("Peter Parker", "040237508");

    school.hireLecturer(lecturer);
    school.addCourse(course, lecturer);

    bool test64 = student.enrollCourse(course);
//    test64 -> false (expected)
    if (!test64)
//        printStudents() print student(s) enrolled in course.
        course.printStudents();

//    Expected Result: The enrollCourse method cannot function
//    because the student to be enrolled in the course is not
//    registered at the school. Returns false, no students are
//    printed to the screen.
//    ##################################################################
```

Case 65: Enrolling to the Course by Student Object (the student is already enrolled the Course)

```
//    ##################################################################
//    Case 65: Enrolling to the Course by Student Object
//    (the student is already enrolled the Course)
//    ------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);
//    create new Student object
```

```
    Student student("Peter Parker", "040237508");

    school.hireLecturer(lecturer);
    school.registerStudent(student);
    school.addCourse(course, lecturer);
    student.enrollCourse(course);

    bool test65 = student.enrollCourse(course);
//    test64 -> false (expected)
    if (!test65)
//        printStudents() print student(s) enrolled in course.
        course.printStudents();

//    Expected Result: The enrollCourse method cannot function
//    because the student to be enrolled in the course is already
//    registered in the course. Returns false, the student who is
//    already enrolled in the course is printed on the screen.
//    ###############################################################
```

Case 66: Enrolling to the Course by Student Object (the course at full capacity)

```
//    ###############################################################
//    Case 66: Enrolling to the Course by Student Object
//    (the course at full capacity)
//    ---------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);
//    create new Student object
    Student student("Peter Parker", "040237550");
    Student student1("Emily Johnson", "040237501");
    Student student2("Michael Brown", "040237502");
    Student student3("Emma Williams", "040237503");
    Student student4("Daniel Jones", "040237504");
    Student student5("Sophia Davis", "040237505");
    Student student6("Matthew Wilson", "040237506");
    Student student7("Olivia Taylor", "040237507");
    Student student8("James Martinez", "040237508");
    Student student9("Isabella Lopez", "040237509");
    Student student10("William Anderson", "040237510");

    school.hireLecturer(lecturer);
```

```
    school.registerStudent(student);
    school.registerStudent(student1);
    school.registerStudent(student2);
    school.registerStudent(student3);
    school.registerStudent(student4);
    school.registerStudent(student5);
    school.registerStudent(student6);
    school.registerStudent(student7);
    school.registerStudent(student8);
    school.registerStudent(student9);
    school.registerStudent(student10);
    school.addCourse(course, lecturer);
    student1.enrollCourse(course);
    student2.enrollCourse(course);
    student3.enrollCourse(course);
    student4.enrollCourse(course);
    student5.enrollCourse(course);
    student6.enrollCourse(course);
    student7.enrollCourse(course);
    student8.enrollCourse(course);
    student9.enrollCourse(course);
    student10.enrollCourse(course);

    bool test66 = student.enrollCourse(course);
//    test66 -> false (expected)
    if (!test66)
//        printStudents() print student(s) enrolled in course.
        course.printStudents();

//    Expected Result: The enrollCourse method cannot function
//    because the course capacity has reached its maximum (10
//    students) and the student (Peter Parker) could not be
//    registered in this case. Returns false, 10 students
//    registered for the course are printed on the screen.
//    ##################################################################
```

Case 67: Enrolling to the Course by Student Object (the course is not active)

```
//    ##################################################################
//    Case 67: Enrolling to the Course by Student Object
//    (the course is not active)
//    ------------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
```

```
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);
//    create new Student object
    Student student("Peter Parker", "040237508");

    school.hireLecturer(lecturer);
    school.registerStudent(student);
    school.addCourse(course, lecturer);
    student.enrollCourse(course);
    school.removeCourse(course);

    bool test67 = student.unEnrollCourse(course);
//    test67 -> false (expected)
    if (!test67)
//        printStudents() print student(s) enrolled in course.
        course.printStudents();

//    Expected Result: The enrollCourse method cannot function;
//    the course to be registered for has been removed by the
//    school (it may not have been opened at all), meaning it is
//    not active. Returns false, no students are printed on the
//    screen.
//    ###############################################################
```

Case 68: Unenrolling to the Course by Student Object

```
//    ###############################################################
//    Case 68: UnEnrolling to the Course by Student Object
//    ---------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);
//    create new Student object
    Student student("Peter Parker", "040237508");

    school.hireLecturer(lecturer);
    school.registerStudent(student);
    school.addCourse(course, lecturer);
    student.enrollCourse(course);

    bool test68 = student.unEnrollCourse(course);
//    test68 -> true (expected)
```

```
    if (test68)
//        printStudents() print student(s) enrolled in course.
        course.printStudents();

//    Expected Result: The unEnrollCourse method performs its
//    function and deletes the student's registration from the
//    course in which he was previously enrolled (it performs
//    the same function as the removeStudentFromCourse() method
//    of the School class and the removeStudent() method of the
//    Course class). It returns true and the student is not
//    displayed on the screen because the student has been
//    unregistered from the course.
//    ################################################################
```

Case 69: Unenrolling to the Course by Student Object (the student is not registered in school)

```
//    ################################################################
//    Case 69: UnEnrolling to the Course by Student Object
//    (the student is not registered in school)
//    ----------------------------------------------------------------
//    create new School object
    School school("Warsaw University of Technology");
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);
//    create new Student object
    Student student("Peter Parker", "040237508");

    school.hireLecturer(lecturer);
    school.registerStudent(student);
    school.addCourse(course, lecturer);
    student.enrollCourse(course);
    student.unregisterFromSchool();

    bool test69 = student.unEnrollCourse(course);
//    test69 -> false (expected)
    if (!test69)
//        printStudents() print student(s) enrolled in course.
        course.printStudents();

//    Expected Result: The unEnrollCourse method cannot function
//    because the student who wants to unEnroll from the course is
//    not registered at the school (maybe deleted or never
//    registered). It returns false and no students are printed to
```

```
//     the screen.
//     ##################################################################
```

Case 70: Unenrolling to the Course by Student Object (the student is not already enrolled the Course)

```
//     ##################################################################
//     Case 70: UnEnrolling to the Course by Student Object
//     (the student is not already enrolled the Course)
//     ------------------------------------------------------------------
//     create new School object
    School school("Warsaw University of Technology");
//     create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//     create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);
//     create new Student object
    Student student("Peter Parker", "040237508");

    school.hireLecturer(lecturer);
    school.registerStudent(student);
    school.addCourse(course, lecturer);

    bool test70 = student.unEnrollCourse(course);
//     test70 -> false (expected)
    if (!test70)
//         printStudents() print student(s) enrolled in course.
        course.printStudents();

//     Expected Result: The unEnrollCourse method cannot fulfill its
//     function because the student who wants to unEnroll from the
//     course is not already enrolled in the course. It returns
//     false and no students are printed to the screen.
//     ##################################################################
```

Case 71: Unenrolling to the Course by Student Object (the course is not active)

```
//     ##################################################################
//     Case 71: UnEnrolling to the Course by Student Object
//     (the course is not active)
//     ------------------------------------------------------------------
//     create new School object
    School school("Warsaw University of Technology");
```

```
//    create new Lecturer object
    Lecturer lecturer("Tony Stark", "Docent", "Electronics");
//    create new Course object
    Course course("Introduction to Building Iron Man", "IBIM", 4);
//    create new Student object
    Student student("Peter Parker", "040237508");

    school.hireLecturer(lecturer);
    school.registerStudent(student);
    school.addCourse(course, lecturer);
    student.enrollCourse(course);
    school.removeCourse(course);

    bool test71 = student.unEnrollCourse(course);
//    test71 -> true (expected)
    if (!test71)
//        printStudents() print student(s) enrolled in course.
        course.printStudents();

//    Expected Result: The unEnrollCourse method cannot fulfill its
//    function because the relevant course has been removed by the
//    school (it may not have been opened at all), that is, it is
//    not active. It is not possible to enroll or cancel
//    registration for an inactive course. It returns false and no
//    students are printed to the screen.
//    ###############################################################
```

About Error Situations (Returning False Value)

As explained in the test programs above, the methods return false when they cannot fulfill their function. While this is happening, the program also prints an error message about the error encountered (cerr << "Opps. There is a glitch..."). Afterwards, the function returns false. So, the program does not stop working. It works as described in the header files and programs above.