|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

**ФАКУЛЬТЕТ** ***ИУК «Информатика и управление»***

**КАФЕДРА** \_\_***ИУК4 «Программное обеспечение ЭВМ, информационные технологии»***

**ЛАБОРАТОРНАЯ РАБОТА №4**

**«Обработка исключений»**

**ДИСЦИПЛИНА: «Объектно-ориентированное программирование»**

|  |  |  |
| --- | --- | --- |
| Выполнил: студент гр. ИУК4-42Б | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Карельский М.К. )  (Подпись) |
| Проверил: | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Былинка М.И. )  (Подпись) |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | | |
|  | - Балльная оценка:  - Оценка: | |

Калуга, 2022

**Цель:** приобрести навыки использования механизм исключений для сигнализации об ошибочных ситуациях и их обработки.

**Задачи:**

1. Научиться использовать систему обработки исключений в C#.
2. Доработать предметную модель из предыдущей лабораторной работы для обработки ошибок.

**Вариант №7**



Рис. 1. Учебные сертификаты

Сущность AttendedCourse может находиться в одном из следующих состояний: «обучение», «аттестация», «отчислен», «архив».

Сразу после создания сущность AttendedCourse находится в состоянии «обучение». В ходе обучения студента заполняется информация о результатах сдачи им требуемых работ.

Периодически преподаватель осуществляет контроль сдачи работ. Если оказывается, что студент на момент контроля не сдал ни одной работы из запланированных, преподаватель отчисляет студента. Сущность AttendedCourse переходит в состояние «отчислен», в ней фиксируется дата/время отчисления.

После сдачи студентом всех работ (включая итоговый экзамен) сущность AttendedCourse переходит в состояние «аттестация». В этом состоянии преподаватель может корректировать результаты студента (но не может добавлять новые записи). Преподаватель принимает решение о выдаче или не выдаче сертификата, фиксирует принятое решение, а также дату/время. Сущность переходит в состояние «архив».

В состоянии «архив» информация редактироваться не может.

В рамках данной лабораторной работы необходимо внести следующие изменения в модель:

* добавить в сущность AttendedCourse атрибуты, содержащие: текущее состояние, дату/время отчисления, дату/время принятия решения о выдаче сертификата, само решение о выдаче сертификата;
* реализовать операции перехода AttendedCourse по состояниям.

Кроме того, необходимо для всех операций модели реализовать проверку предусловий:

* корректности переданных параметров;
* допустимости выполнения операции в текущем состоянии объекта.

**Листинг:**

***AttendedCourse.cs***

using System;

using System.Collections.Generic;

namespace LW\_2.AttendedCourse

{

public abstract class AttendedCourse

{

public enum Status

{

Study,

Certification,

Expelled,

Archive

}

public AttendedCourse(DateTimeOffset enrollmentDate, Student student, Course course)

{

EnrollmentDate = enrollmentDate;

Student = student;

Course = course;

Results = new List<CourseWorkResult>();

State = Status.Study;

ExpellingDate = null;

ArchiveDate = null;

IsCertificated = false;

}

public bool TryToCreateResult(DateTimeOffset dateTime, bool isPassed, ushort score, CourseWorkRequirements work)

{

if (State != Status.Study)

return false;

CreateResult(dateTime, isPassed, score, work);

return true;

}

protected abstract void CreateResult(DateTimeOffset dateTime, bool isPassed, ushort score, CourseWorkRequirements work);

public abstract bool IsPassed();

public bool TryToExpell(DateTimeOffset time)

{

if (State != Status.Study)

return false;

foreach (var result in Results)

if (result.IsPassed)

return false;

foreach (var work in Course.CourseWorks)

if (work.Deadline < time)

{

State = Status.Expelled;

ExpellingDate = time;

return true;

}

return false;

}

public bool TryToStartCertification()

{

if (!IsPassed() || State != Status.Study)

return false;

State = Status.Certification;

return true;

}

public bool TryToChangeResultPassing(int index, bool isPassed)

{

if (State != Status.Certification)

return false;

Results[index].IsPassed = isPassed;

return true;

}

public bool TryToChangeResultScore(int index, ushort score)

{

if (State != Status.Certification)

return false;

Results[index].Score = score;

return true;

}

public bool TryToArchive(DateTimeOffset time, bool isCertificated)

{

if (State != Status.Certification)

return false;

ArchiveDate = time;

IsCertificated = isCertificated;

State = Status.Archive;

return true;

}

public DateTimeOffset EnrollmentDate { get; }

public string RegNumber { get { return Student.RegNumber; } }

public Student Student { get; }

public Course Course { get; }

public List<CourseWorkResult> Results { get; }

public Status State { get; private set; }

public DateTimeOffset? ExpellingDate { get; private set; }

public DateTimeOffset? ArchiveDate { get; private set; }

public bool IsCertificated { get; private set; }

}

}

***Course.cs***

using System;

using System.Collections.Generic;

using System.Text.RegularExpressions;

namespace LW\_2

{

public class Course

{

public Course(uint code, string title, string professor, ushort passingScore, Rule ruleType)

{

if (!Regex.IsMatch(title.ToLower(), @"[a-z]+"))

throw new ArgumentException("Title without letters");

if (!Regex.IsMatch(professor.ToLower(), @"[a-z]+"))

throw new ArgumentException("Professor's name without letters");

Code = code;

Title = title;

Professor = professor;

PassingScore = passingScore;

CourseWorks = new List<CourseWorkRequirements>();

RuleType = ruleType;

}

public enum Rule

{

OfSeventyFive,

OfFifty,

OfOneTime,

OfThreeTimes,

OfUnlimitedTimes,

OfVariousTimes

}

public AttendedCourse.AttendedCourse EnrollStudent(

DateTimeOffset enrollmentDate, Student student)

{

return RuleType switch

{

Rule.OfSeventyFive => new AttendedCourse.AttendedCourseWithRuleOfSeventyFive(enrollmentDate, student, this),

Rule.OfFifty => new AttendedCourse.AttendedCourseWithRuleOfFifty(enrollmentDate, student, this),

Rule.OfOneTime => new AttendedCourse.AttendedCourseWithRuleOfOneTime(enrollmentDate, student, this),

Rule.OfThreeTimes => new AttendedCourse.AttendedCourseWithRuleOfThreeTimes(enrollmentDate, student, this),

Rule.OfUnlimitedTimes => new AttendedCourse.AttendedCourseWithRuleOfUnlimitedTimes(enrollmentDate, student, this),

Rule.OfVariousTimes => new AttendedCourse.AttendedCourseWithRuleOfVariousTimes(enrollmentDate, student, this),

\_ => throw new NotImplementedException()

};

}

public void CreateWork(CourseWorkRequirements.WorkType type, string title, ushort maxScore, DateTimeOffset deadline)

{

CourseWorks.Add(new CourseWorkRequirements(type, title, maxScore, deadline));

}

public uint Code { get; }

public string Title { get; }

public string Professor { get; }

public ushort PassingScore { get; }

public List<CourseWorkRequirements> CourseWorks { get; }

public Rule RuleType { get; }

}

}

***CourseWorkRequirements.cs***

using System;

using System.Text.RegularExpressions;

namespace LW\_2

{

public class CourseWorkRequirements

{

public CourseWorkRequirements(WorkType type, string title, ushort maxScore, DateTimeOffset deadline)

{

if (!Regex.IsMatch(title.ToLower(), @"[a-z]+"))

throw new ArgumentException("Title without letters");

if (maxScore == 0)

throw new ArgumentException("Max score can't be equal to 0");

Type = type;

Title = title;

MaxScore = maxScore;

Deadline = deadline;

}

public enum WorkType

{

Test,

Homework,

Exam

}

public WorkType Type { get; }

public string Title { get; }

public ushort MaxScore { get; }

public DateTimeOffset Deadline { get; }

}

}

***Student.cs***

using System;

using System.Text.RegularExpressions;

namespace LW\_2

{

public class Student

{

public Student() : this("", "", DateTimeOffset.Parse("01/01/01")) { }

public Student(string regNumber, string fullName) : this(regNumber, fullName, DateTimeOffset.Parse("01/01/01")) { }

public Student(string regNumber, string fullName, DateTimeOffset birthDate)

{

if (!Regex.IsMatch(fullName.ToLower(), @"[a-z]+"))

throw new ArgumentException("Name without letters");

RegNumber = regNumber;

FullName = fullName;

BirthDate = birthDate;

}

public string RegNumber { get; }

public string FullName { get; }

public DateTimeOffset BirthDate { get; }

}

}

***UnitTest.cs***

using Microsoft.VisualStudio.TestTools.UnitTesting;

using LW\_2.AttendedCourse;

using LW\_2;

using System;

namespace Testing

{

[TestClass]

public class Testing

{

[TestMethod]

public void TestExpelledState()

{

Student mike = new Student("2", "Mike", DateTimeOffset.Parse("02.02.2002"));

Course course = new Course(1, "Math", "Ivanov I.I.", 80, Course.Rule.OfFifty);

course.CreateWork(CourseWorkRequirements.WorkType.Homework, "Homework", 70, DateTimeOffset.Parse("25.05.2022"));

course.CreateWork(CourseWorkRequirements.WorkType.Exam, "Exam", 30, DateTimeOffset.Parse("30.05.2022"));

AttendedCourse mikeCourse = course.EnrollStudent(DateTimeOffset.Parse("19.05.2022"), mike);

mikeCourse.TryToExpell(DateTimeOffset.Parse("27.05.2022"));

Assert.AreEqual(false, mikeCourse.TryToCreateResult(DateTimeOffset.Parse("27.05.2022"), true, 50, mikeCourse.Course.CourseWorks[0]));

Assert.AreEqual(false, mikeCourse.TryToExpell(DateTimeOffset.Parse("29.05.2022")));

Assert.AreEqual(false, mikeCourse.TryToStartCertification());

Assert.AreEqual(false, mikeCourse.TryToChangeResultPassing(0, false));

Assert.AreEqual(false, mikeCourse.TryToChangeResultScore(0, 25));

Assert.AreEqual(false, mikeCourse.TryToArchive(DateTimeOffset.Parse("01.06.2022"), false));

}

[TestMethod]

public void TestStudyState()

{

Student bob = new Student("1", "Bob", DateTimeOffset.Parse("01.01.2001"));

Student mike = new Student("2", "Mike", DateTimeOffset.Parse("02.02.2002"));

Course course = new Course(1, "Math", "Ivanov I.I.", 80, Course.Rule.OfFifty);

course.CreateWork(CourseWorkRequirements.WorkType.Homework, "Homework", 70, DateTimeOffset.Parse("25.05.2022"));

course.CreateWork(CourseWorkRequirements.WorkType.Exam, "Exam", 30, DateTimeOffset.Parse("30.05.2022"));

AttendedCourse bobCourse = course.EnrollStudent(DateTimeOffset.Parse("19.05.2022"), bob);

AttendedCourse mikeCourse = course.EnrollStudent(DateTimeOffset.Parse("19.05.2022"), mike);

Assert.AreEqual(true, bobCourse.TryToCreateResult(DateTimeOffset.Parse("23.05.2022"), true, 70, bobCourse.Course.CourseWorks[0]));

Assert.AreEqual(false, bobCourse.TryToChangeResultScore(0, 30));

Assert.AreEqual(false, bobCourse.TryToChangeResultPassing(0, false));

Assert.AreEqual(false, bobCourse.TryToArchive(DateTimeOffset.Parse("01.06.2022"), true));

Assert.AreEqual(false, bobCourse.TryToExpell(DateTimeOffset.Parse("27.05.2022")));

Assert.AreEqual(true, mikeCourse.TryToExpell(DateTimeOffset.Parse("27.05.2022")));

Assert.AreEqual(AttendedCourse.Status.Expelled, mikeCourse.State);

Assert.AreEqual(false, bobCourse.TryToStartCertification());

bobCourse.TryToCreateResult(DateTimeOffset.Parse("30.05.2022"), true, 25, bobCourse.Course.CourseWorks[1]);

Assert.AreEqual(true, bobCourse.TryToStartCertification());

}

[TestMethod]

public void TestCertificationState()

{

Student bob = new Student("1", "Bob", DateTimeOffset.Parse("01.01.2001"));

Course course = new Course(1, "Math", "Ivanov I.I.", 80, Course.Rule.OfFifty);

course.CreateWork(CourseWorkRequirements.WorkType.Homework, "Homework", 70, DateTimeOffset.Parse("25.05.2022"));

course.CreateWork(CourseWorkRequirements.WorkType.Exam, "Exam", 30, DateTimeOffset.Parse("30.05.2022"));

AttendedCourse bobCourse = course.EnrollStudent(DateTimeOffset.Parse("19.05.2022"), bob);

bobCourse.TryToCreateResult(DateTimeOffset.Parse("23.05.2022"), true, 70, bobCourse.Course.CourseWorks[0]);

bobCourse.TryToCreateResult(DateTimeOffset.Parse("30.05.2022"), true, 25, bobCourse.Course.CourseWorks[1]);

bobCourse.TryToStartCertification();

Assert.AreEqual(false, bobCourse.TryToCreateResult(DateTimeOffset.Parse("01.06.2022"), true, 30, bobCourse.Course.CourseWorks[1]));

Assert.AreEqual(false, bobCourse.TryToExpell(DateTimeOffset.Parse("01.06.2022")));

Assert.AreEqual(false, bobCourse.TryToStartCertification());

Assert.AreEqual(true, bobCourse.TryToChangeResultScore(0, 30));

Assert.AreEqual(true, bobCourse.TryToChangeResultPassing(0, false));

Assert.AreEqual(true, bobCourse.TryToArchive(DateTimeOffset.Parse("01.06.2022"), true));

Assert.AreEqual(AttendedCourse.Status.Archive, bobCourse.State);

Assert.AreEqual(true, bobCourse.IsCertificated);

}

[TestMethod]

public void TestArchiveState()

{

Student bob = new Student("1", "Bob", DateTimeOffset.Parse("01.01.2001"));

Course course = new Course(1, "Math", "Ivanov I.I.", 80, Course.Rule.OfFifty);

course.CreateWork(CourseWorkRequirements.WorkType.Homework, "Homework", 70, DateTimeOffset.Parse("25.05.2022"));

course.CreateWork(CourseWorkRequirements.WorkType.Exam, "Exam", 30, DateTimeOffset.Parse("30.05.2022"));

AttendedCourse bobCourse = course.EnrollStudent(DateTimeOffset.Parse("19.05.2022"), bob);

bobCourse.TryToCreateResult(DateTimeOffset.Parse("23.05.2022"), true, 70, bobCourse.Course.CourseWorks[0]);

bobCourse.TryToCreateResult(DateTimeOffset.Parse("30.05.2022"), true, 25, bobCourse.Course.CourseWorks[1]);

bobCourse.TryToStartCertification();

bobCourse.TryToArchive(DateTimeOffset.Parse("01.06.2022"), true);

Assert.AreEqual(false, bobCourse.TryToCreateResult(DateTimeOffset.Parse("01.06.2022"), true, 30, bobCourse.Course.CourseWorks[1]));

Assert.AreEqual(false, bobCourse.TryToExpell(DateTimeOffset.Parse("01.06.2022")));

Assert.AreEqual(false, bobCourse.TryToStartCertification());

Assert.AreEqual(false, bobCourse.TryToChangeResultScore(0, 30));

Assert.AreEqual(false, bobCourse.TryToChangeResultPassing(0, false));

Assert.AreEqual(false, bobCourse.TryToArchive(DateTimeOffset.Parse("02.06.2022"), false));

}

}

}

**Результат:**

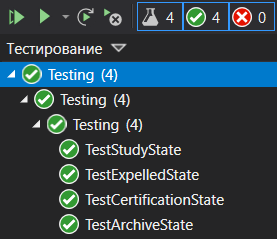


Рис. 2. Результат

**Вывод:** в ходе выполнения лабораторной работы были получены навыки использования механизма исключений в C#.