|  |  |
| --- | --- |
| **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 №3**

**«Наследование и полиморфизм»**

**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 механизмов наследования и полиморфизма при реализации объектно-ориентированной модели.

**Задачи:**

1. Научиться применять принципы SOLID на практике.
2. Доработать предметную модель из предыдущей лабораторной работы.

**Вариант №7**



Рис. 1. Учебные сертификаты

Каждый профессор сам определяет требования, которые необходимо выполнить, чтобы получить сертификат о прохождении курса. В настоящее время для разных курсов действуют следующие правила:

* правила, реализованные в лабораторной работе № 2;
* успешно принятые работы не могут пересдаваться; если работа сдана не в срок, то при подсчете учитываются только 50% набранных за нее баллов (округление вниз);
* все работы могут пересдаваться только один раз (кроме экзамена – экзамен пересдавать нельзя), при этом всегда учитывается последний результат; сдача работ с опозданием не штрафуется (кроме экзамена – экзамен может быть сдан только в срок);
* все работы могут пересдаваться до трех раз, при этом всегда учитывается максимальный результат; сдача работ с опозданием на 1 неделю не штрафуется; при сдаче с опозданием более 1 недели при подсчете учитываются только 80% набранных за работу баллов;
* все работы могут пересдаваться неограниченное количество раз, при этом всегда учитывается последний результат; при сдаче с опозданием не более 1 недели при подсчете учитываются только 85% набранных за работу баллов, работы с опозданием более 1 недели не принимаются;
* домашние задания могут пересдаваться до пяти раз; рубежные тесты и экзамен – до трех раз; при подсчете учитывается только 90% максимального результата (если была хотя бы одна пересдача); если работа сдана не в срок, то при подсчете учитываются только: 80% набранных баллов для домашнего задания, 75% баллов для рубежного теста; 60% баллов для экзамена (округление вверх).

Необходимо модифицировать модель, реализованную в лабораторной работе № 2, так, чтобы полученное решение учитывало новые правила выдачи сертификатов. При этом решение должно быть таким, что добавление новых правил не должно приводить к переписыванию существующего кода.

**Листинг:**

***AttendedCourse.cs***

using System;

using System.Collections.Generic;

namespace LW\_2.AttendedCourse

{

public abstract class AttendedCourse

{

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

{

EnrollmentDate = enrollmentDate;

Student = student;

Course = course;

Results = new List<CourseWorkResult>();

}

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

public abstract bool IsPassed();

public DateTimeOffset EnrollmentDate { get; }

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

public Student Student { get; }

public Course Course { get; }

public List<CourseWorkResult> Results { get; }

}

}

***AttendedCourseWithRuleOfFifty.cs***

using System;

namespace LW\_2.AttendedCourse

{

public class AttendedCourseWithRuleOfFifty: AttendedCourse

{

public AttendedCourseWithRuleOfFifty(DateTimeOffset enrollmentDate,

Student student, Course course) : base(enrollmentDate, student, course) { }

public override void CreateResult(DateTimeOffset dateTime, bool isPassed, ushort score, CourseWorkRequirements work)

{

foreach (CourseWorkResult result in Results)

if (result.Requirements == work)

if (result.IsPassed)

return;

if (dateTime > work.Deadline)

score = (ushort)(score \* 0.5);

Results.Add(new CourseWorkResult(dateTime, isPassed, score, work));

}

public override bool IsPassed()

{

ushort sum = 0;

foreach (CourseWorkResult result in Results)

if (result.IsPassed)

sum += result.Score;

return sum >= Course.PassingScore;

}

}

}

***AttendedCourseWithRuleOfOneTime.cs***

using System;

using System.Collections.Generic;

namespace LW\_2.AttendedCourse

{

class AttendedCourseWithRuleOfOneTime : AttendedCourse

{

public AttendedCourseWithRuleOfOneTime(DateTimeOffset enrollmentDate,

Student student, Course course) : base(enrollmentDate, student, course) { }

public override void CreateResult(DateTimeOffset dateTime, bool isPassed, ushort score, CourseWorkRequirements work)

{

if (work.Type == CourseWorkRequirements.WorkType.Exam)

{

foreach (CourseWorkResult result in Results)

if (result.Requirements == work)

return;

if (work.Deadline < dateTime)

return;

}

else

{

int number = 0;

foreach (CourseWorkResult result in Results)

if (result.Requirements == work)

++number;

if (number > 1)

return;

}

Results.Add(new CourseWorkResult(dateTime, isPassed, score, work));

}

public override bool IsPassed()

{

List<CourseWorkResult> finalResults = new List<CourseWorkResult>();

foreach (CourseWorkResult result in Results)

{

bool isFound = false;

for (int i = 0; i < finalResults.Count; ++i)

if (result.Requirements == finalResults[i].Requirements)

{

isFound = true;

if (result.DateTime > finalResults[i].DateTime)

finalResults[i] = result;

}

if (!isFound)

finalResults.Add(result);

}

ushort sum = 0;

foreach (CourseWorkResult result in finalResults)

if (result.IsPassed)

sum += result.Score;

return sum >= Course.PassingScore;

}

}

}

***AttendedCourseWithRuleOfSeventyFive.cs***

using System;

namespace LW\_2.AttendedCourse

{

public class AttendedCourseWithRuleOfSeventyFive : AttendedCourse

{

public AttendedCourseWithRuleOfSeventyFive(DateTimeOffset enrollmentDate,

Student student, Course course) : base(enrollmentDate, student, course) { }

public override void CreateResult(DateTimeOffset dateTime, bool isPassed, ushort score, CourseWorkRequirements work)

{

foreach (CourseWorkResult result in Results)

if (result.Requirements == work)

if (result.IsPassed)

return;

if (dateTime > work.Deadline)

score = (ushort)(score \* 0.75);

Results.Add(new CourseWorkResult(dateTime, isPassed, score, work));

}

public override bool IsPassed()

{

ushort sum = 0;

foreach (CourseWorkResult result in Results)

if (result.IsPassed)

sum += result.Score;

return sum >= Course.PassingScore;

}

}

}

***AttendedCourseWithRuleOfThreeTimes.cs***

using System;

using System.Collections.Generic;

namespace LW\_2.AttendedCourse

{

class AttendedCourseWithRuleOfThreeTimes : AttendedCourse

{

public AttendedCourseWithRuleOfThreeTimes(DateTimeOffset enrollmentDate,

Student student, Course course) : base(enrollmentDate, student, course) { }

public override void CreateResult(DateTimeOffset dateTime, bool isPassed, ushort score, CourseWorkRequirements work)

{

int number = 0;

foreach (CourseWorkResult result in Results)

if (result.Requirements == work)

++number;

if (number > 3)

return;

if ((dateTime - work.Deadline).TotalDays > 7)

score = (ushort)(score \* 0.8);

Results.Add(new CourseWorkResult(dateTime, isPassed, score, work));

}

public override bool IsPassed()

{

List<CourseWorkResult> finalResults = new List<CourseWorkResult>();

foreach (CourseWorkResult result in Results)

{

bool isFound = false;

for (int i = 0; i < finalResults.Count; ++i)

if (result.Requirements == finalResults[i].Requirements)

{

isFound = true;

if (result.Score > finalResults[i].Score)

finalResults[i] = result;

}

if (!isFound)

finalResults.Add(result);

}

ushort sum = 0;

foreach (CourseWorkResult result in finalResults)

if (result.IsPassed)

sum += result.Score;

return sum >= Course.PassingScore;

}

}

}

***AttendedCourseWithRuleOfUnlimitedTimes.cs***

using System;

using System.Collections.Generic;

namespace LW\_2.AttendedCourse

{

class AttendedCourseWithRuleOfUnlimitedTimes : AttendedCourse

{

public AttendedCourseWithRuleOfUnlimitedTimes(DateTimeOffset enrollmentDate,

Student student, Course course) : base(enrollmentDate, student, course) { }

public override void CreateResult(DateTimeOffset dateTime, bool isPassed, ushort score, CourseWorkRequirements work)

{

double lateness = (dateTime - work.Deadline).TotalDays;

if (lateness >= 0 && lateness <= 7)

score = (ushort)(score \* 0.85);

else if (lateness > 7)

return;

Results.Add(new CourseWorkResult(dateTime, isPassed, score, work));

}

public override bool IsPassed()

{

List<CourseWorkResult> finalResults = new List<CourseWorkResult>();

foreach (CourseWorkResult result in Results)

{

bool isFound = false;

for (int i = 0; i < finalResults.Count; ++i)

if (result.Requirements == finalResults[i].Requirements)

{

isFound = true;

if (result.DateTime > finalResults[i].DateTime)

finalResults[i] = result;

}

if (!isFound)

finalResults.Add(result);

}

ushort sum = 0;

foreach (CourseWorkResult result in finalResults)

if (result.IsPassed)

sum += result.Score;

return sum >= Course.PassingScore;

}

}

}

***AttendedCourseWithRuleOfVariousTimes.cs***

using System;

using System.Collections.Generic;

namespace LW\_2.AttendedCourse

{

class AttendedCourseWithRuleOfVariousTimes : AttendedCourse

{

public AttendedCourseWithRuleOfVariousTimes(DateTimeOffset enrollmentDate,

Student student, Course course) : base(enrollmentDate, student, course) { }

public override void CreateResult(DateTimeOffset dateTime, bool isPassed, ushort score, CourseWorkRequirements work)

{

int number = 0;

foreach (CourseWorkResult result in Results)

if (result.Requirements == work)

++number;

if (work.Type == CourseWorkRequirements.WorkType.Homework)

{

if (number > 5)

return;

}

else if (number > 3)

return;

if (dateTime > work.Deadline)

if (work.Type == CourseWorkRequirements.WorkType.Homework)

score = (ushort)Math.Ceiling(score \* 0.8);

else if (work.Type == CourseWorkRequirements.WorkType.Test)

score = (ushort)Math.Ceiling(score \* 0.75);

else

score = (ushort)Math.Ceiling(score \* 0.6);

Results.Add(new CourseWorkResult(dateTime, isPassed, score, work));

}

private struct ResultRetakePair

{

public ResultRetakePair(CourseWorkResult result, bool isRetaken)

{

this.result = result;

this.isRetaken = isRetaken;

}

public CourseWorkResult result;

public bool isRetaken;

}

public override bool IsPassed()

{

List<ResultRetakePair> finalResults = new List<ResultRetakePair>();

foreach (CourseWorkResult result in Results)

{

bool isFound = false;

for (int i = 0; i < finalResults.Count; ++i)

if (result.Requirements == finalResults[i].result.Requirements)

{

isFound = true;

ResultRetakePair pair = finalResults[i];

pair.isRetaken = true;

if (result.Score > finalResults[i].result.Score)

pair.result = result;

finalResults[i] = pair;

}

if (!isFound)

finalResults.Add(new ResultRetakePair(result, false));

}

ushort sum = 0;

foreach (ResultRetakePair pair in finalResults)

if (pair.result.IsPassed)

if (pair.isRetaken)

sum += (ushort)(pair.result.Score \* 0.9);

else

sum += pair.result.Score;

return sum >= Course.PassingScore;

}

}

}

***Course.cs***

using System;

using System.Collections.Generic;

namespace LW\_2

{

public class Course

{

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

{

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;

namespace LW\_2

{

public class CourseWorkRequirements

{

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

{

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; }

}

}

***CourseWorkResult.cs***

using System;

namespace LW\_2

{

public class CourseWorkResult

{

public CourseWorkResult(DateTimeOffset dateTime, bool isPassed, ushort score, CourseWorkRequirements requirements)

{

DateTime = dateTime;

IsPassed = isPassed;

if (score > requirements.MaxScore)

Score = requirements.MaxScore;

else

Score = score;

Requirements = requirements;

}

public DateTimeOffset DateTime { get; }

public bool IsPassed { get; }

public ushort Score { get; }

public CourseWorkRequirements Requirements { get; }

}

}

***Student.cs***

namespace LW\_2

{

public class Student

{

public Student(uint regNumber, string fullName)

{

RegNumber = regNumber;

FullName = fullName;

}

public uint RegNumber { get; }

public string FullName { get; }

}

}

***UnitTest.cs***

using Microsoft.VisualStudio.TestTools.UnitTesting;

using LW\_2;

using System;

namespace Testing

{

[TestClass]

public class Testing

{

[TestMethod]

public void TestRuleOfSeventyFive()

{

Student student = new Student(1, "Ivan");

Course course = new Course(123, "Math", "Ramazanov A.K.", 70, Course.Rule.OfSeventyFive);

course.CreateWork(CourseWorkRequirements.WorkType.Homework, "DZ", 100, DateTimeOffset.Parse("10/03/2022"));

LW\_2.AttendedCourse.AttendedCourse attendedCourse = course.EnrollStudent(DateTimeOffset.Parse("01/03/2022"), student);

attendedCourse.CreateResult(DateTimeOffset.Parse("05/03/2022"), false, 50, course.CourseWorks[0]);

Assert.AreEqual(50, attendedCourse.Results[0].Score);

attendedCourse.CreateResult(DateTimeOffset.Parse("15/03/2022"), true, 100, course.CourseWorks[0]);

Assert.AreEqual(75, attendedCourse.Results[1].Score);

attendedCourse.CreateResult(DateTimeOffset.Parse("17/03/2022"), true, 100, course.CourseWorks[0]);

Assert.AreEqual(2, attendedCourse.Results.Count);

}

[TestMethod]

public void TestRuleOfFifty()

{

Student student = new Student(1, "Ivan");

Course course = new Course(123, "Math", "Ramazanov A.K.", 70, Course.Rule.OfFifty);

course.CreateWork(CourseWorkRequirements.WorkType.Homework, "DZ", 100, DateTimeOffset.Parse("10/03/2022"));

LW\_2.AttendedCourse.AttendedCourse attendedCourse = course.EnrollStudent(DateTimeOffset.Parse("01/03/2022"), student);

attendedCourse.CreateResult(DateTimeOffset.Parse("05/03/2022"), false, 50, course.CourseWorks[0]);

Assert.AreEqual(50, attendedCourse.Results[0].Score);

attendedCourse.CreateResult(DateTimeOffset.Parse("15/03/2022"), true, 99, course.CourseWorks[0]);

Assert.AreEqual(49, attendedCourse.Results[1].Score);

attendedCourse.CreateResult(DateTimeOffset.Parse("17/03/2022"), true, 100, course.CourseWorks[0]);

Assert.AreEqual(2, attendedCourse.Results.Count);

}

[TestMethod]

public void TestRuleOfOneTime()

{

Student student = new Student(1, "Ivan");

Course course = new Course(123, "Math", "Ramazanov A.K.", 70, Course.Rule.OfOneTime);

course.CreateWork(CourseWorkRequirements.WorkType.Homework, "DZ", 100, DateTimeOffset.Parse("10/03/2022"));

course.CreateWork(CourseWorkRequirements.WorkType.Exam, "Exam", 100, DateTimeOffset.Parse("20/03/2022"));

LW\_2.AttendedCourse.AttendedCourse attendedCourse = course.EnrollStudent(DateTimeOffset.Parse("01/03/2022"), student);

attendedCourse.CreateResult(DateTimeOffset.Parse("5/03/2022"), true, 80, course.CourseWorks[0]);

attendedCourse.CreateResult(DateTimeOffset.Parse("12/03/2022"), true, 40, course.CourseWorks[0]);

Assert.AreEqual(false, attendedCourse.IsPassed());

attendedCourse.CreateResult(DateTimeOffset.Parse("14/03/2022"), true, 60, course.CourseWorks[0]);

attendedCourse.CreateResult(DateTimeOffset.Parse("25/03/2022"), true, 60, course.CourseWorks[1]);

Assert.AreEqual(2, attendedCourse.Results.Count);

attendedCourse.CreateResult(DateTimeOffset.Parse("15/03/2022"), true, 60, course.CourseWorks[1]);

attendedCourse.CreateResult(DateTimeOffset.Parse("17/03/2022"), true, 60, course.CourseWorks[1]);

Assert.AreEqual(3, attendedCourse.Results.Count);

}

[TestMethod]

public void TestRuleOfThreeTimes()

{

Student student = new Student(1, "Ivan");

Course course = new Course(123, "Math", "Ramazanov A.K.", 70, Course.Rule.OfThreeTimes);

course.CreateWork(CourseWorkRequirements.WorkType.Homework, "DZ", 100, DateTimeOffset.Parse("10/03/2022"));

LW\_2.AttendedCourse.AttendedCourse attendedCourse = course.EnrollStudent(DateTimeOffset.Parse("01/03/2022"), student);

attendedCourse.CreateResult(DateTimeOffset.Parse("5/03/2022"), true, 80, course.CourseWorks[0]);

attendedCourse.CreateResult(DateTimeOffset.Parse("15/03/2022"), true, 40, course.CourseWorks[0]);

Assert.AreEqual(true, attendedCourse.IsPassed());

Assert.AreEqual(80, attendedCourse.Results[0].Score);

Assert.AreEqual(40, attendedCourse.Results[1].Score);

attendedCourse.CreateResult(DateTimeOffset.Parse("17/03/2022"), true, 50, course.CourseWorks[0]);

Assert.AreEqual(50, attendedCourse.Results[2].Score);

attendedCourse.CreateResult(DateTimeOffset.Parse("18/03/2022"), true, 100, course.CourseWorks[0]);

Assert.AreEqual(80, attendedCourse.Results[3].Score);

attendedCourse.CreateResult(DateTimeOffset.Parse("20/03/2022"), true, 100, course.CourseWorks[0]);

Assert.AreEqual(4, attendedCourse.Results.Count);

}

[TestMethod]

public void TestRuleOfUnlimitedTimes()

{

Student student = new Student(1, "Ivan");

Course course = new Course(123, "Math", "Ramazanov A.K.", 70, Course.Rule.OfUnlimitedTimes);

course.CreateWork(CourseWorkRequirements.WorkType.Homework, "DZ", 100, DateTimeOffset.Parse("10/03/2022"));

LW\_2.AttendedCourse.AttendedCourse attendedCourse = course.EnrollStudent(DateTimeOffset.Parse("01/03/2022"), student);

attendedCourse.CreateResult(DateTimeOffset.Parse("5/03/2022"), true, 80, course.CourseWorks[0]);

attendedCourse.CreateResult(DateTimeOffset.Parse("7/03/2022"), true, 40, course.CourseWorks[0]);

Assert.AreEqual(false, attendedCourse.IsPassed());

Assert.AreEqual(40, attendedCourse.Results[1].Score);

attendedCourse.CreateResult(DateTimeOffset.Parse("15/03/2022"), true, 100, course.CourseWorks[0]);

Assert.AreEqual(85, attendedCourse.Results[2].Score);

attendedCourse.CreateResult(DateTimeOffset.Parse("17/03/2022"), true, 100, course.CourseWorks[0]);

Assert.AreEqual(4, attendedCourse.Results.Count);

attendedCourse.CreateResult(DateTimeOffset.Parse("13/04/2022"), true, 100, course.CourseWorks[0]);

Assert.AreEqual(4, attendedCourse.Results.Count);

}

[TestMethod]

public void TestRuleOfVariousTimes()

{

Student student = new Student(1, "Ivan");

Course course\_1 = new Course(111, "Math", "Ramazanov A.K.", 85, Course.Rule.OfVariousTimes);

course\_1.CreateWork(CourseWorkRequirements.WorkType.Exam, "Exam", 100, DateTimeOffset.Parse("10/03/2022"));

LW\_2.AttendedCourse.AttendedCourse attendedCourse\_1 = course\_1.EnrollStudent(DateTimeOffset.Parse("01/03/2022"), student);

attendedCourse\_1.CreateResult(DateTimeOffset.Parse("5/03/2022"), true, 90, course\_1.CourseWorks[0]);

Assert.AreEqual(true, attendedCourse\_1.IsPassed());

attendedCourse\_1.CreateResult(DateTimeOffset.Parse("6/03/2022"), true, 87, course\_1.CourseWorks[0]);

Assert.AreEqual(false, attendedCourse\_1.IsPassed());

attendedCourse\_1.CreateResult(DateTimeOffset.Parse("7/03/2022"), true, 100, course\_1.CourseWorks[0]);

Assert.AreEqual(true, attendedCourse\_1.IsPassed());

attendedCourse\_1.CreateResult(DateTimeOffset.Parse("8/03/2022"), true, 100, course\_1.CourseWorks[0]);

attendedCourse\_1.CreateResult(DateTimeOffset.Parse("9/03/2022"), true, 100, course\_1.CourseWorks[0]);

Assert.AreEqual(4, attendedCourse\_1.Results.Count);

Course course\_2 = new Course(222, "English", "Vlasko N.K.", 85, Course.Rule.OfVariousTimes);

course\_2.CreateWork(CourseWorkRequirements.WorkType.Exam, "Exam", 100, DateTimeOffset.Parse("10/03/2022"));

LW\_2.AttendedCourse.AttendedCourse attendedCourse\_2 = course\_2.EnrollStudent(DateTimeOffset.Parse("01/03/2022"), student);

attendedCourse\_2.CreateResult(DateTimeOffset.Parse("15/03/2022"), true, 100, course\_2.CourseWorks[0]);

Assert.AreEqual(false, attendedCourse\_2.IsPassed());

Course course\_3 = new Course(333, "LITA", "Nikitenko U.V.", 85, Course.Rule.OfVariousTimes);

course\_3.CreateWork(CourseWorkRequirements.WorkType.Homework, "DZ", 100, DateTimeOffset.Parse("10/03/2022"));

LW\_2.AttendedCourse.AttendedCourse attendedCourse\_3 = course\_3.EnrollStudent(DateTimeOffset.Parse("01/03/2022"), student);

attendedCourse\_3.CreateResult(DateTimeOffset.Parse("15/03/2022"), true, 100, course\_3.CourseWorks[0]);

Assert.AreEqual(false, attendedCourse\_3.IsPassed());

attendedCourse\_3.CreateResult(DateTimeOffset.Parse("21/03/2022"), true, 100, course\_3.CourseWorks[0]);

attendedCourse\_3.CreateResult(DateTimeOffset.Parse("22/03/2022"), true, 100, course\_3.CourseWorks[0]);

attendedCourse\_3.CreateResult(DateTimeOffset.Parse("23/03/2022"), true, 100, course\_3.CourseWorks[0]);

attendedCourse\_3.CreateResult(DateTimeOffset.Parse("24/03/2022"), true, 100, course\_3.CourseWorks[0]);

attendedCourse\_3.CreateResult(DateTimeOffset.Parse("25/03/2022"), true, 100, course\_3.CourseWorks[0]);

attendedCourse\_3.CreateResult(DateTimeOffset.Parse("26/03/2022"), true, 100, course\_3.CourseWorks[0]);

Assert.AreEqual(6, attendedCourse\_3.Results.Count);

Course course\_4 = new Course(444, "TISD", "Pchelinceva N.I.", 85, Course.Rule.OfVariousTimes);

course\_4.CreateWork(CourseWorkRequirements.WorkType.Test, "Test", 100, DateTimeOffset.Parse("10/03/2022"));

LW\_2.AttendedCourse.AttendedCourse attendedCourse\_4 = course\_4.EnrollStudent(DateTimeOffset.Parse("01/03/2022"), student);

attendedCourse\_4.CreateResult(DateTimeOffset.Parse("15/03/2022"), true, 100, course\_4.CourseWorks[0]);

Assert.AreEqual(false, attendedCourse\_4.IsPassed());

attendedCourse\_4.CreateResult(DateTimeOffset.Parse("21/03/2022"), true, 100, course\_4.CourseWorks[0]);

attendedCourse\_4.CreateResult(DateTimeOffset.Parse("22/03/2022"), true, 100, course\_4.CourseWorks[0]);

attendedCourse\_4.CreateResult(DateTimeOffset.Parse("23/03/2022"), true, 100, course\_4.CourseWorks[0]);

attendedCourse\_4.CreateResult(DateTimeOffset.Parse("24/03/2022"), true, 100, course\_4.CourseWorks[0]);

Assert.AreEqual(4, attendedCourse\_4.Results.Count);

}

}

}

**Результат:**

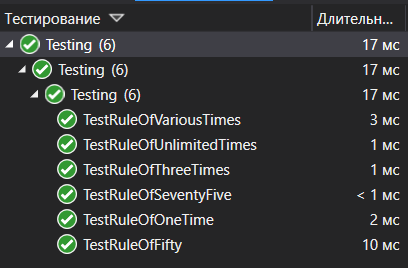


Рис. 2. Результат

**Вывод:** в ходе выполнения лабораторной работы были получены практические навыки реализации наследования и полиморфизма, модификации модели.