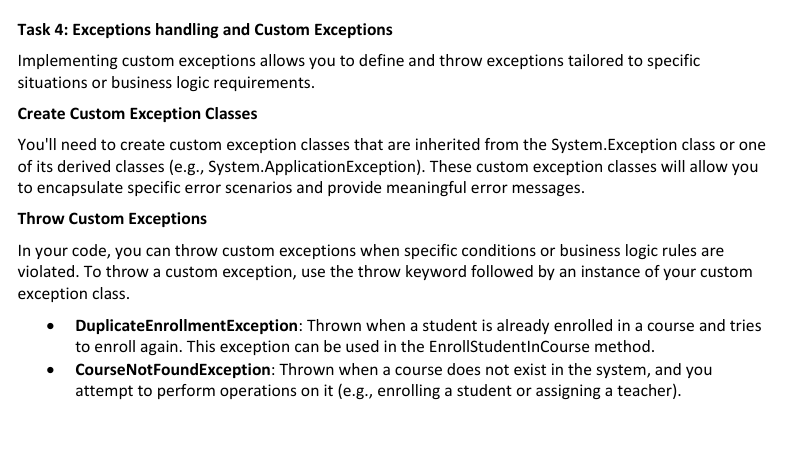
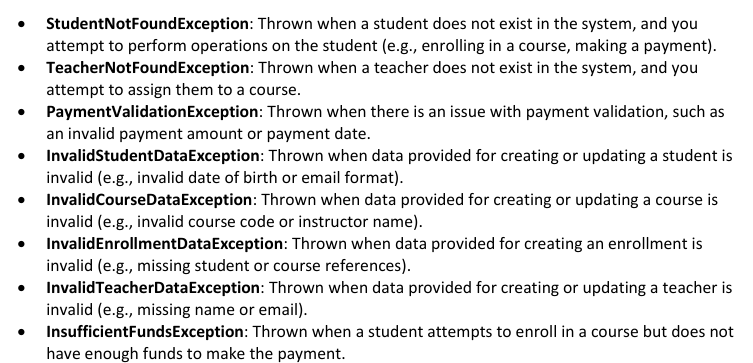
**Task 4 Questions**





**Output Screenshots are at the end in Task\_4 class**

**The Exception Class**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SIS.CoustomExceptions

{

public class DuplicateEnrollmentException : Exception

{

public DuplicateEnrollmentException(string message) : base(message) {}

}

public class CourseNotFoundException : Exception

{

public CourseNotFoundException(string message) : base(message) { }

}

public class StudentNotFoundException : Exception

{

public StudentNotFoundException(string message) : base(message) { }

}

public class TeacherNotFoundException : Exception

{

public TeacherNotFoundException(string message) : base(message) { }

}

public class PaymentValidationException : Exception

{

public PaymentValidationException(string message) : base(message) { }

}

public class InvalidStudentDataException : Exception

{

public InvalidStudentDataException(string message) : base(message) { }

}

public class InvalidCourseDataException : Exception

{

public InvalidCourseDataException(string message) : base(message) { }

}

public class InvalidEnrollmentDataException : Exception

{

public InvalidEnrollmentDataException(string message) : base(message) { }

}

public class InvalidTeacherDataException : Exception

{

public InvalidTeacherDataException(string message) : base(message) { }

}

public class InsufficientFundsException : Exception

{

public InsufficientFundsException(string message) : base(message) { }

}

}

**The Checking Class [For Throwing exceptions]**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SIS.CoustomExceptions

{

internal class Checking

{

public static void CheckDuplicateEnrollment(Student student,Course course)

{

foreach(Enrollment enrollment in student.EnrolledCourses)

{

if(enrollment.CourseID == course)

{

throw new DuplicateEnrollmentException($"Student {student.FirstName} is already enrolled in {course.CourseName}");

}

}

}

public static void CheckCourseExists(Course course)

{

if (course == null)

{

throw new CourseNotFoundException("Course reference is null");

}

bool exist = Course.AllCourses.Contains(course);

if (!exist)

{

throw new CourseNotFoundException($"Course {course.CourseName} does not exist in the system");

}

}

public static void CheckStudentExists(Student student)

{

bool exists = Student.AllStudents.Contains(student);

if (!exists)

{

throw new StudentNotFoundException($"Student {student.FirstName} {student.LastName} does not exist in the system");

}

}

public static void CheckTeacherExists(Teacher teacher)

{

if (!Teacher.AllTeachers.Contains(teacher))

{

throw new TeacherNotFoundException($"Teacher {teacher.FirstName} {teacher.LastName} does not exist in the system");

}

}

public static void CheckPaymentValid(decimal amount, DateTime paymentDate)

{

if (amount <= 0)

{

throw new PaymentValidationException("Payment amount must be greater than zero");

}

if (paymentDate > DateTime.Now)

{

throw new PaymentValidationException("Payment date cannot be in the future");

}

if (paymentDate < new DateTime(2024, 12, 1))

{

throw new PaymentValidationException("Payment date must be after 1st December 2024");

}

}

public static void CheckStudentDataValid(string firstName, string lastName, DateTime dateOfBirth, string email, string phoneNumber)

{

if (string.IsNullOrWhiteSpace(firstName) || string.IsNullOrWhiteSpace(lastName))

throw new InvalidStudentDataException("Name cannot be empty");

if (dateOfBirth >= DateTime.Now)

throw new InvalidStudentDataException("Date of birth must be in the past");

if (!email.Contains("@") || !email.Contains("."))

throw new InvalidStudentDataException("Email format is invalid");

if (phoneNumber.Length != 10 || !phoneNumber.All(char.IsDigit))

throw new InvalidStudentDataException("Phone number must be 10 digits long");

}

public static void CheckCourseDataValid(string courseCode, string courseName)

{

if (string.IsNullOrWhiteSpace(courseCode) || courseCode.Length < 4)

throw new InvalidCourseDataException("Course code must be at least 4 characters long");

if (string.IsNullOrWhiteSpace(courseName))

throw new InvalidCourseDataException("Course name cannot be empty");

}

public static void CheckCourseDataValid(string courseCode, string courseName, string instructor)

{

CheckCourseDataValid(courseCode, courseName); // Reuse

if (string.IsNullOrWhiteSpace(instructor))

throw new InvalidCourseDataException("Instructor name cannot be empty");

}

public static void CheckEnrollmentDataValid(Student student, Course course)

{

if (student == null)

throw new InvalidEnrollmentDataException("Student reference cannot be null.");

if (course == null)

throw new InvalidEnrollmentDataException("Course reference cannot be null.");

}

public static void ValidateTeacherData(string firstName, string lastName, string email)

{

if (string.IsNullOrWhiteSpace(firstName) || string.IsNullOrWhiteSpace(lastName) || string.IsNullOrWhiteSpace(email))

{

throw new InvalidTeacherDataException("Teacher must have a valid first name, last name, and email.");

}

}

public static void ValidateTeacherUpdate(string name, string email)

{

if (string.IsNullOrWhiteSpace(name) || !name.Contains(" ") || string.IsNullOrWhiteSpace(email))

{

throw new InvalidTeacherDataException("Update failed: Name must contain both first and last name, and email must not be empty.");

}

}

public static void CheckSufficientFunds(decimal amount)

{

if (amount < 40000)

{

throw new InsufficientFundsException($"Payment of {amount} is insufficient, Minimum requirement is 40000 for initial payment");

}

}

}

}

**The Task\_4 Class [for triggering the exceptions]**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SIS.Task

{

internal class Task\_4

{

Student student1 = new Student(101, "Caitlyn", "Kiraman", new DateTime(2000, 5, 15), "cait@mail.com", "1234567890");

Student student2 = new Student(102, "David", "Johnson", new DateTime(1999, 8, 22), "dave@mail.com", "9876543210");

Student student3 = new Student(107, "April", "Smith", new DateTime(2000, 10, 1), "april@mail.com", "9591529583");

Student student4 = new Student(100, "Beth", "Michel", new DateTime(1999, 5, 12), "beth@mail.com", "9754283655");

Student dummystud = new Student(190, "dummy", "dummy", new DateTime(1999, 5, 12), "dummy@mail.com", "9754283655");

Course math = new Course(201, "Algebra", "MATH101", "");

Course bio = new Course(201, "Biology", "BIO401", "");

Course history = new Course(201, "History", "HIS74", "");

Course dummyCourse = new Course(909, "Dummy", "DC90", "");

Teacher teacher1 = new Teacher(1, "Jayce", "Norten", "jay@mail.com", "Algebra");

Teacher teacher2 = new Teacher(2, "Haley", "Fields", "haley@mail.com", "History");

Teacher teacher3 = new Teacher(3, "Roy", "Wells", "roy@mail.com", "Biology");

Teacher dummyTeacher = new Teacher(4, "dummy", "dummy", "dummy@mail.com", "Biology");

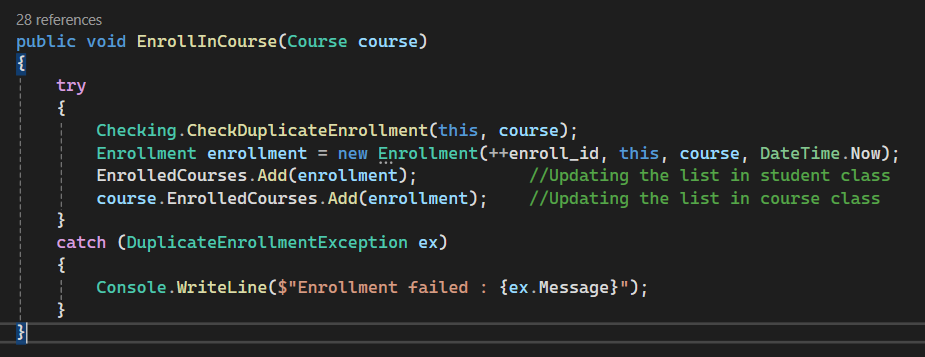
public void Exception1()

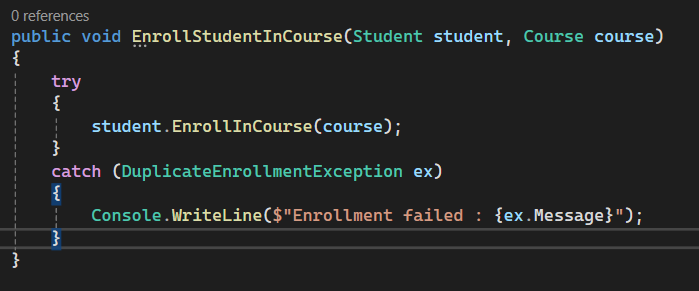
{

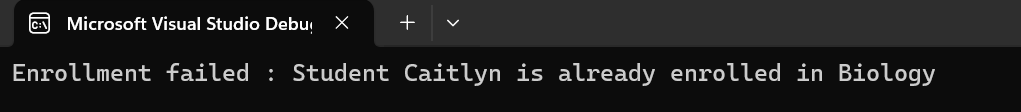
student1.EnrollInCourse(bio);

student1.EnrollInCourse(bio);

}







public void Exception2()

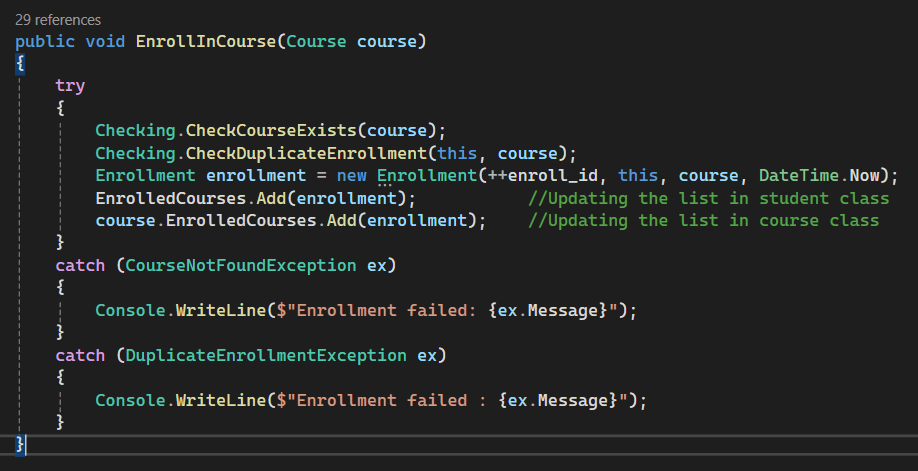
{

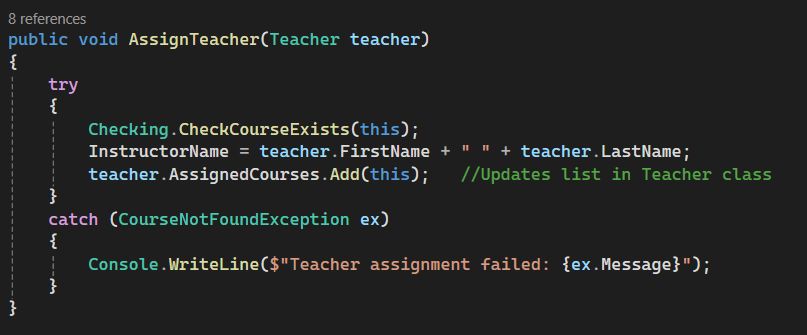
Course.AllCourses.Remove(dummyCourse);

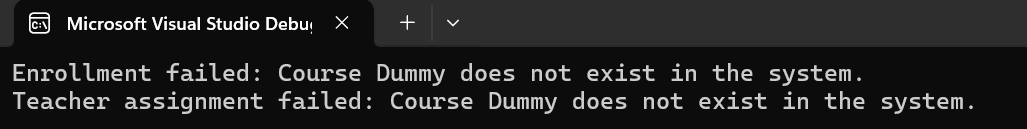
student1.EnrollInCourse(dummyCourse);

dummyCourse.AssignTeacher(teacher1);

}







public void Exception3()

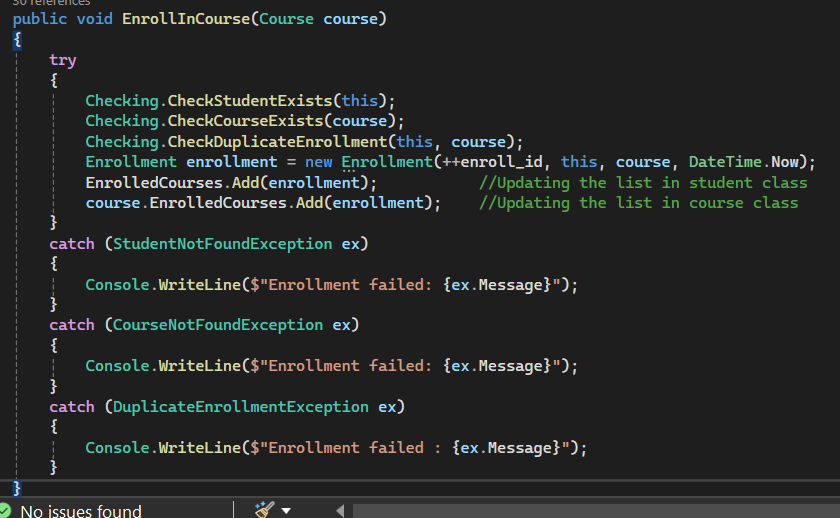
{

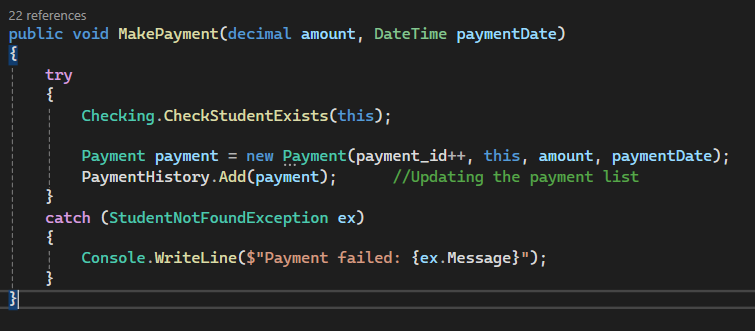
Student.AllStudents.Remove(dummystud);

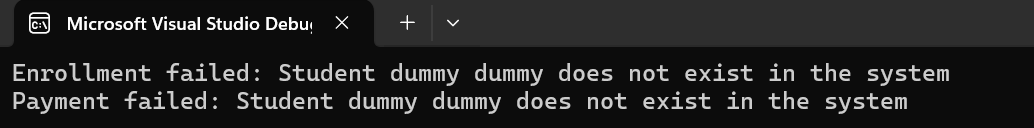
dummystud.EnrollInCourse(bio);

dummystud.MakePayment(6000, new DateTime(2025, 2, 15));

}







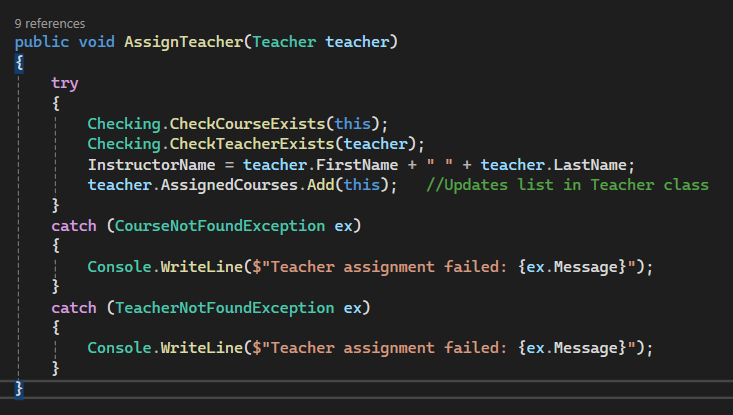
public void Exception4()

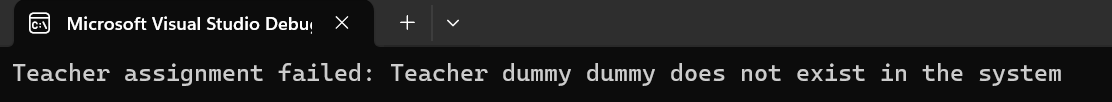
{

Teacher.AllTeachers.Remove(dummyTeacher);

bio.AssignTeacher(dummyTeacher);

}





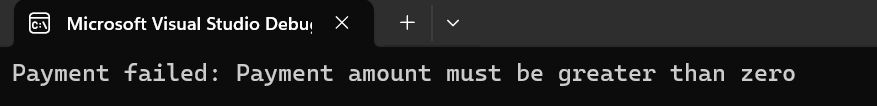
public void Exception5()

{

student2.MakePayment(-20, new DateTime(2027, 3, 3));

}



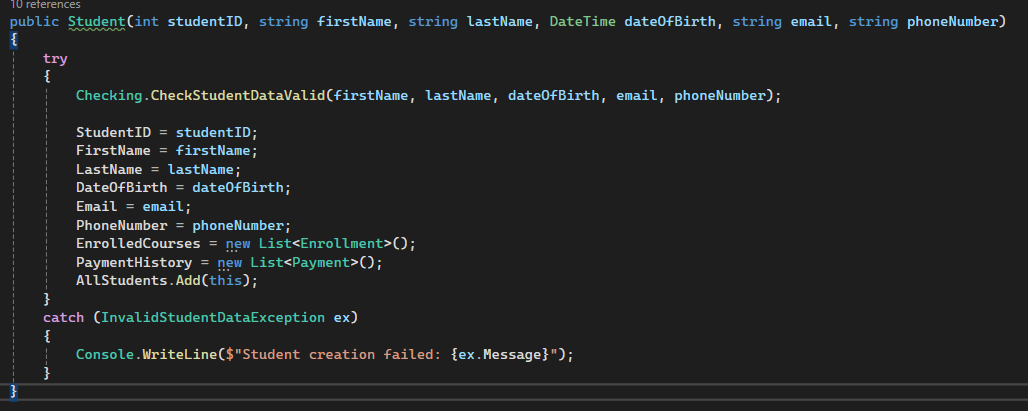


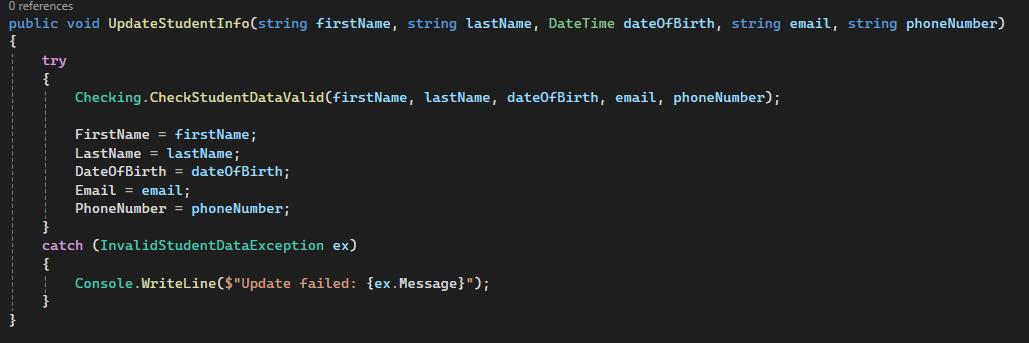
public void Exception6()

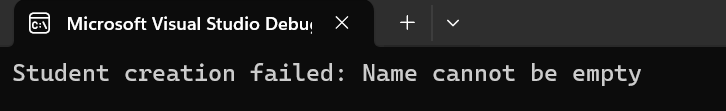
{

Student student4 = new Student(100, " ", "Michel", new DateTime(1999, 5, 12), "beth@mail.com", "9754283655");

}





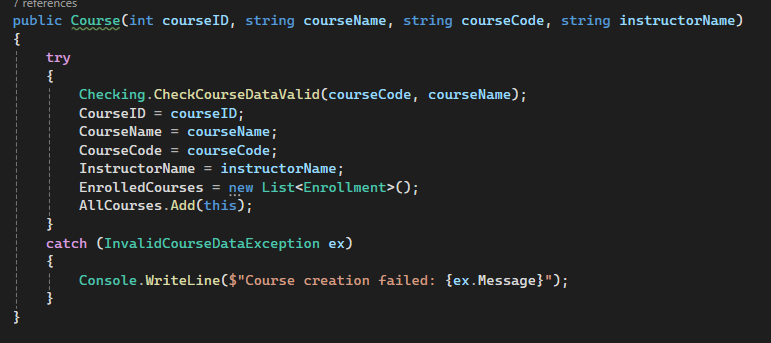


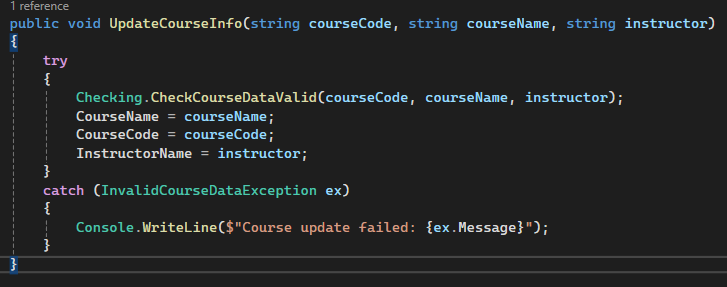
public void Exception7()

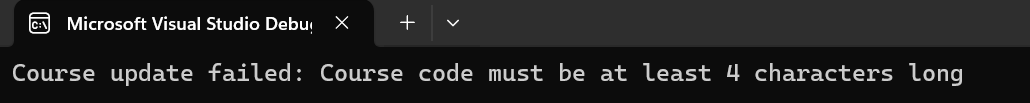
{

math.UpdateCourseInfo("", "", "");

}





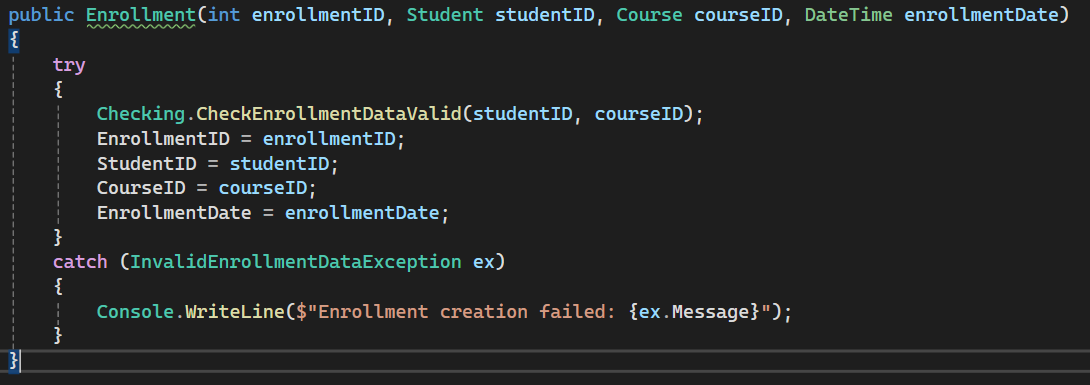


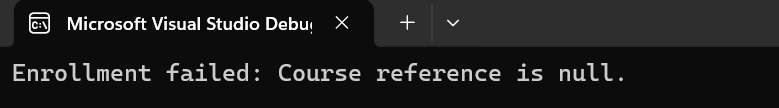
public void Exception8()

{

student1.EnrollInCourse(null);

}



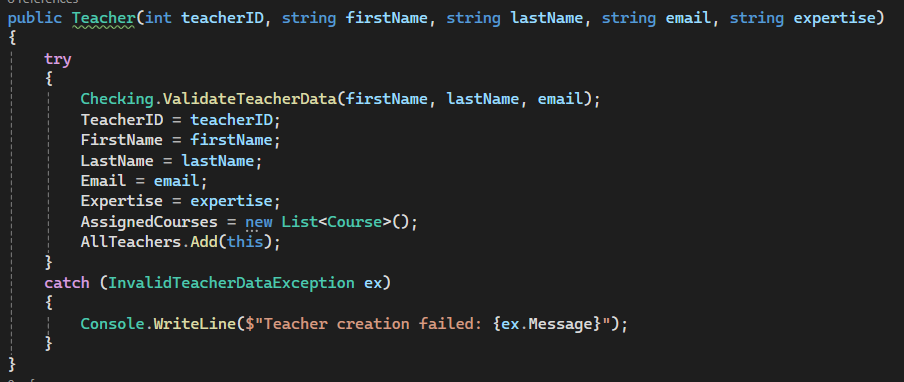


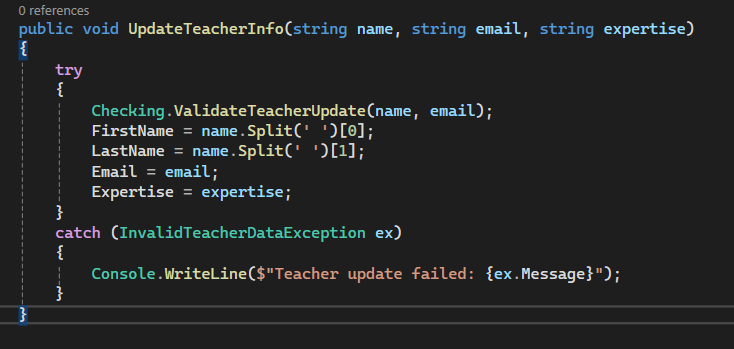
public void Exception9()

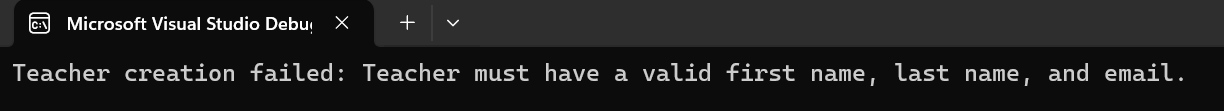
{

Teacher teacher3 = new Teacher(3, "", "Wells", "roy@mail.com", "Biology");

}





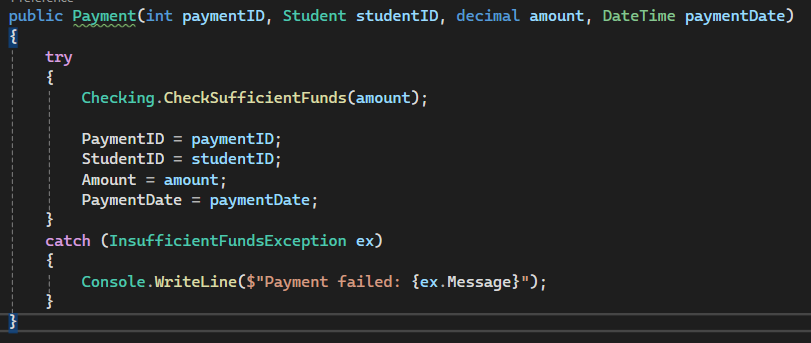


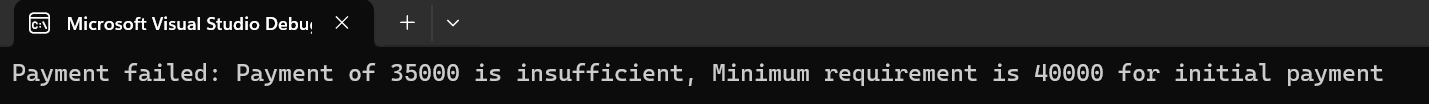
public void Exception10()

{

student1.MakePayment(35000, new DateTime(2025, 2, 2));

}





}

}