МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Інститут комп'ютерних наук та інформаційних технологій
Кафедра програмного забезпечення

ЗВІТ

До лабораторної роботи №2

На тему: «Linq, List and Dictionary»

З дисципліни: *«Моделювання та аналіз програмного забезпечення»*

Лектор:
доцент кафедри ПЗ
Сердюк П. В.

Виконав:
ст. групи ПЗ-22
Павлів М. Я.

Прийняв:
асист. кафедри ПЗ
Шкраб Р. Р.
«___» ____ 2022р.
Σ= ____ ……

Львів – 2022

Тема роботи: Linq, List and Dictionary.

Мета роботи: Робота з масивами та структурами List, Dictionary. Використання технології LINQ.

Індивідуальне завдання

Повинні бути реалізовані:

1. Селекція частини інформації (Select).
2. Вибірка певної інформації (Where)
3. Операції як з списком List так і з словником Dictionary
4. Реалізувати власні методи розширювання.
5. Показати використання анонімних класів та ініціалізаторів
6. Відсортувати за якимось критерієм використовуючи шаблон IComparer.
7. Конвертувати списки в масив.
8. Відсортувати масив/список за ім’ям чи за кількістю елементів.

Хід роботи

1. Показав використання Select.

```
public static IEnumerable<T> Get<T>(this Repository repository, Func<Course, T> selector)
{
    return repository.Courses.Select(selector);
}

public static IEnumerable<T> Get<T>(this Repository repository, Func<Event, T> selector)
{
    return repository.Events.Select(selector);
}

public static IEnumerable<T> Get<T>(this Repository repository, Func<Organizer, T> selector)
{
    return repository.Organizers.Select(selector);
}

public static IEnumerable<T> Get<T>(this Repository repository, Func<Student, T> selector)
{
    return repository.Students.Select(selector);
}
```

2. Показав використання Where.

```
public static IEnumerable<Course> GetCourses(this Repository repository, Func<Course, bool> selector)
{
    return repository.Courses.Where(selector);
}

public static IEnumerable<Event> GetEvents(this Repository repository, Func<Event, bool> selector)
{
    return repository.Events.Where(selector);
}

public static IEnumerable<Organizer> GetOrganizers(this Repository repository, Func<Organizer, bool> selector)
{
    return repository.Organizers.Where(selector);
}

public static IEnumerable<Student> GetStudents(this Repository repository, Func<Student, bool> selector)
{
    return repository.Students.Where(selector);
}
```

3. Показав операції з List та Dictionary.

```
public IEnumerable<Student> this[Organizer organizer]
{
    get
    {
        var obj = _studentsByOrganizer
            .Select((kv, i) => new { Organizer = kv.Key, Index = i })
            .FirstOrDefault(obj => obj.Organizer.Equals(organizer));

        return obj is null ? Enumerable.Empty<Student>() : _studentsByOrganizer.ElementAt(obj.Index).Value;
    }
}
```

```
public void Add(Course course)
{
    _courses.Add(course);
}

public void Add(Event e)
{
    _events.Add(e);
}

public void Add(Organizer organizer)
{
    _organizers.Add(organizer);

    if (_studentsByOrganizer.TryAdd(organizer, organizer.SubscribedStudents) == false)
    {
        throw new InvalidOperationException();
    }

    foreach (var student in organizer.SubscribedStudents)
    {
        _students.Add(student);
    }
}

public void Add(Student student)
{
    _students.Add(student);
}
```

```
public void Update(Organizer organizer)
{
    if (_studentsByOrganizer.ContainsKey(organizer) == false)
    {
        throw new InvalidOperationException();
    }

    _studentsByOrganizer[organizer] = organizer.SubscribedStudents;
}
```

4. Реалізував методи розширювання.
5. Показав використання анонімних класів та ініціалізаторів.

```
public IEnumerable<Student> this[Organizer organizer]
{
    get
    {
        var obj = _studentsByOrganizer
            .Select((kv, i) => new { Organizer = kv.Key, Index = i })
            .FirstOrDefault(obj => obj.Organizer.Equals(organizer));

        return obj is null ? Enumerable.Empty<Student>() : _studentsByOrganizer.ElementAt(obj.Index).Value;
    }
}
```

```
_repository = new Repository
{
    Courses = new List<Course>
    {
        new Course(
            "React course.",
            "Learn React from zero to hero.",
            DateTime.Today.AddDays(1),
            DateTime.Today.AddDays(31)
        ),

        new Course(
            "C# course",
            "Learn C# from zero to hero",
            DateTime.Today.AddDays(8),
            DateTime.Today.AddDays(38)
        )
    }
};
```

6. Відсортував список, використовуючи IComparer.

```
Student[] sortedStudents = repository.Sort(new MyComparer());
```

```
public Student[] Sort(IComparer<Student> studentComparer)
{
    var studentsList = _students.ToList();
    studentsList.Sort(studentComparer);
    return studentsList.ToArray();
}
```

7. Конвертував список в масив.
8. Відсортував список за вказаним критерієм.

```
public IEnumerable<Course> OrderBy<TKey>(Func<Course, TKey> order)
{
    return _courses.OrderBy(order);
}

public IEnumerable<Event> OrderBy<TKey>(Func<Event, TKey> order)
{
    return _events.OrderBy(order);
}
```

```
var orderedCourses = repository.OrderBy(course => course.StartDate);
var expectedOrderedCourses = new List<Course>
```

```
var orderedEvents = repository.OrderBy((Event e) => e.Title);
var expectedOrderedEvents = new List<Event>
```

Код програми

Course.cs

```
namespace Events;

public class Course
{
    public string Title { get; }

    public string Description { get; }

    public DateTime StartDate { get; }

    public DateTime EndDate { get; }

    public Course(string title, string description, DateTime start, DateTime end)
    {
        if (String.IsNullOrWhiteSpace(title))
        {
            throw new ArgumentException("Title must be non-empty.", nameof(title));
        }

        if (String.IsNullOrWhiteSpace(description))
        {
            throw new ArgumentException("Description must be non-empty.",
nameof(description));
        }

        if (end < start)
        {
            throw new ArgumentException("End date cannot be less than start date.");
        }

        Title = title;
        Description = description;
        StartDate = start;
        EndDate = end;
    }

    public override bool Equals(object? obj)
    {
        if (obj is null or not Course)
        {
            return false;
        }
```

```
        var course = obj as Course;

        return course.Description.Equals(this.Description) &&
course.Title.Equals(this.Title);
    }

    public override int GetHashCode()
    {
        return HashCode.Combine(Title, Description, EndDate, StartDate);
    }
}
```

Event.cs

```
namespace Events;

public class Event
{
    public string Title { get; }
    public string Description { get; }
    public Organizer Organizer { get; }

    public Event(string title, string description, Organizer organizer)
    {
        Title = title;
        Description = description;
        Organizer = organizer;
    }

    public override bool Equals(object? obj)
    {
        if (obj is null or not Event)
        {
            return false;
        }

        Event e = obj as Event;

        return e.Title.Equals(this.Title, StringComparison.InvariantCultureIgnoreCase);
    }

    public override int GetHashCode()
    {
        return HashCode.Combine(Title, Description);
    }
}
```

Organizer.cs

```
namespace Events;

public class Organizer
{
    private readonly List<Student> _subscribedStudents;
    private int _id = Random.Shared.Next(int.MinValue, int.MaxValue);

    public int Id
    {
        get => _id;
        init => _id = value;
    }

    public IEnumerable<Student> SubscribedStudents => _subscribedStudents;

    public Organizer(IEnumerable<Student>? subscriptions = null)
    {
        _subscribedStudents = subscriptions?.ToList() ?? new List<Student>();
    }
```

```
    public void AddSubscription(Student student)
    {
        _subscribedStudents.Add(student);
    }

    public bool RemoveSubscription(Student student)
    {
        return _subscribedStudents.Remove(student);
    }

    public void StartCourse(Course course)
    {
        foreach (var student in _subscribedStudents)
        {
            student.Notify(course);
        }
    }

    public override bool Equals(object? obj)
    {
        if (obj is null or not Organizer)
        {
            return false;
        }

        return ((Organizer)obj).Id == this.Id;
    }

    public override int GetHashCode()
    {
        return Id.GetHashCode();
    }
}
```

Repository.cs

```
namespace Events;

public class Repository
{
    private readonly List<Course> _courses = new();
    private readonly List<Event> _events = new();
    private readonly List<Organizer> _organizers = new();
    private readonly HashSet<Student> _students = new();
    private readonly Dictionary<Organizer, IEnumerable<Student>> _studentsByOrganizer =
new();

    public IEnumerable<Course> Courses
    {
        get => _courses;
        init => _courses = value.ToList();
    }

    public IEnumerable<Event> Events
    {
        get => _events;
        init => _events = value.ToList();
    }

    public IEnumerable<Organizer> Organizers
    {
        get => _organizers;
        init
        {
            _organizers = value.ToList();
            _organizers.ForEach(org =>
```

```
            {
                _studentsByOrganizer.Add(org, org.SubscribedStudents);
                foreach (var student in org.SubscribedStudents)
                {
                    _students.Add(student);
                }
            });
        }
    }

    public IEnumerable<Student> Students
    {
        get => _students;
        init => _students = value.ToHashSet();
    }

    public IEnumerable<Student> this[Organizer organizer]
    {
        get
        {
            var obj = _studentsByOrganizer
                .Select((kv, i) => new { Organizer = kv.Key, Index = i })
                .FirstOrDefault(obj => obj.Organizer.Equals(organizer));

            return obj is null ? Enumerable.Empty<Student>() :
_studentsByOrganizer.ElementAt(obj.Index).Value;
        }
    }

    public void Add(Course course)
    {
        _courses.Add(course);
    }

    public void Add(Event e)
    {
        _events.Add(e);
    }

    public void Add(Organizer organizer)
    {
        _organizers.Add(organizer);

        if (_studentsByOrganizer.TryAdd(organizer, organizer.SubscribedStudents) ==
false)
        {
            throw new InvalidOperationException();
        }

        foreach (var student in organizer.SubscribedStudents)
        {
            _students.Add(student);
        }
    }

    public void Add(Student student)
    {
        _students.Add(student);
    }

    public void Update(Organizer organizer)
    {
        if (_studentsByOrganizer.ContainsKey(organizer) == false)
        {
            throw new InvalidOperationException();
        }
```

```
        _studentsByOrganizer[organizer] = organizer.SubscribedStudents;
    }

    public Student[] Sort(IComparer<Student> studentComparer)
    {
        var studentsList = _students.ToList();
        studentsList.Sort(studentComparer);
        return studentsList.ToArray();
    }

    public IEnumerable<Course> OrderBy<TKey>(Func<Course, TKey> order)
    {
        return _courses.OrderBy(order);
    }

    public IEnumerable<Event> OrderBy<TKey>(Func<Event, TKey> order)
    {
        return _events.OrderBy(order);
    }
}
```

RepositoryExtensions.cs

```
namespace Events;

public static class RepositoryExtensions
{
    public static IEnumerable<Course> GetCourses(this Repository repository, Func<Course,
bool> selector)
    {
        return repository.Courses.Where(selector);
    }

    public static IEnumerable<Event> GetEvents(this Repository repository, Func<Event,
bool> selector)
    {
        return repository.Events.Where(selector);
    }

    public static IEnumerable<Organizer> GetOrganizers(this Repository repository,
Func<Organizer, bool> selector)
    {
        return repository.Organizers.Where(selector);
    }

    public static IEnumerable<Student> GetStudents(this Repository repository,
Func<Student, bool> selector)
    {
        return repository.Students.Where(selector);
    }

    public static IEnumerable<T> Get<T>(this Repository repository, Func<Course, T>
selector)
    {
        return repository.Courses.Select(selector);
    }

    public static IEnumerable<T> Get<T>(this Repository repository, Func<Event, T>
selector)
    {
        return repository.Events.Select(selector);
    }

    public static IEnumerable<T> Get<T>(this Repository repository, Func<Organizer, T>
selector)
    {
```

```
        return repository.Organizers.Select(selector);
    }

    public static IEnumerable<T> Get<T>(this Repository repository, Func<Student, T>
selector)
    {
        return repository.Students.Select(selector);
    }
}
```

Student.cs

```
namespace Events;

public class Student
{
    public string FirstName { get; }
    public string LastName { get; }
    public string Email { get; }

    public string FullName => String.Concat(FirstName, " ", LastName);

    public Student(string firstName, string lastName, string email)
    {
        if (String.IsNullOrWhiteSpace(firstName))
        {
            throw new ArgumentException("First name must be non-empty.",
nameof(firstName));
        }

        if (String.IsNullOrWhiteSpace(lastName))
        {
            throw new ArgumentException("Title must be non-empty.", nameof(lastName));
        }

        FirstName = firstName;
        LastName = lastName;
        Email = email;
    }

    public void Notify(Course course)
    {
      // sending mail to email address
    }

    public override int GetHashCode()
    {
        return HashCode.Combine(Email, FullName);
    }

    public override bool Equals(object? obj)
    {
        if (obj is null)
        {
            return false;
        }

        if (obj is not Student student)
        {
            return false;
        }

        return student.FullName.Equals(this.FullName) &&
student.Email.Equals(this.Email);
    }
}
```

RepositoryTests.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using NUnit.Framework;

namespace Events.Test;

[TestFixture]
public class RepositoryTests
{
    private Repository _repository;

    [SetUp]
    public void Setup()
    {
        _repository = new Repository
        {
            Courses = new List<Course>
            {
                new Course(
                    "React course.",
                    "Learn React from zero to hero.",
                    DateTime.Today.AddDays(1),
                    DateTime.Today.AddDays(31)
                ),

                new Course(
                    "C# course",
                    "Learn C# from zero to hero",
                    DateTime.Today.AddDays(8),
                    DateTime.Today.AddDays(38)
                )
            }
        };
    }

    [Test]
    public void Repository_Courses_Contains2Elements()
    {
        var courses = _repository.Courses;

        Assert.That(courses.Count(), Is.EqualTo(2));
    }

    [Test]
    public void Courses_Elements_AreEqualToExpected()
    {
        var actualCourses = _repository.Courses;
        var expectedCourses = new List<Course>
        {
            new Course(
                "React course.",
                "Learn React from zero to hero.",
                DateTime.Today.AddDays(1),
                DateTime.Today.AddDays(31)
            ),

            new Course(
                "C# course",
                "Learn C# from zero to hero",
                DateTime.Today.AddDays(8),
                DateTime.Today.AddDays(38)
            )
        };

        Assert.That(actualCourses, Is.EquivalentTo(expectedCourses));
```

```
}

[Test]
public void Events_Empty_IsEmpty()
{
    var actualEvents = _repository.Events;

    Assert.That(actualEvents, Is.Empty);
}

[Test]
public void Organizers_Empty_IsEmpty()
{
    var actualOrganizers = _repository.Organizers;

    Assert.That(actualOrganizers, Is.Empty);
}

[Test]
public void Students_Empty_IsEmpty()
{
    var actualStudents = _repository.Students;

    Assert.That(actualStudents, Is.Empty);
}

[Test]
public void Indexer_NonexistentOrganizer_ReturnsEmptyCollection()
{
    var organizer = new Organizer();

    var actualStudents = _repository[organizer];
    Assert.That(actualStudents, Is.Empty);
}

[Test]
public void Update_NonexistentOrganizer_ThrowInvalidOperation()
{
    var organizer = new Organizer();

    TestDelegate code = () => _repository.Update(organizer);

    Assert.Throws<InvalidOperationException>(code);
}

[Test]
public void Add_Course_UpdatesList()
{
    var course = new Course(
        "MAPZ",
        "Become pro with C#!",
        DateTime.Now.AddDays(-60),
        DateTime.Now.AddDays(60)
    );

    _repository.Add(course);

    var actualCourses = _repository.Courses;

    Assert.That(actualCourses.Count(), Is.EqualTo(3));
    Assert.That(actualCourses.Last(), Is.EqualTo(course));
}

[Test]
public void Add_Organizer_UpdatesList()
{
```

```
    var organizer = new Organizer
    {
        Id = 1
    };

    _repository.Add(organizer);
    var first = _repository.Organizers.First();
    var last = _repository.Organizers.Last();
    Assert.That(first, Is.EqualTo(last).And.EqualTo(organizer));
}

[Test]
public void Add_Event_UpdatesList()
{
    var newEvent = new Event(
        "Meeting",
        "Meet and make new friends!",
        new Organizer()
    );

    _repository.Add(newEvent);
    Assert.That(_repository.Events.Last(), Is.EqualTo(newEvent));
    Assert.That(_repository.Events.Count(), Is.EqualTo(1));
}

[Test]
public void Add_Student_UpdatesList()
{
    var student = new Student("Maksym", "Pavliv", "maksym.pavliv.pz.2020@lpnu.ua");

    var prevCount = _repository.Students.Count();
    _repository.Add(student);
    var actualStudents = _repository.Students.ToList();
    Assert.That(actualStudents.Count, Is.EqualTo(prevCount + 1));
    Assert.That(actualStudents.Last(), Is.EqualTo(student));
}

[Test]
public void Sort_4Students_SortsList()
{
    var students = new List<Student>
    {
        new Student("Maksym", "Pavliv", "maksym.pavliv.pz.2020@lpnu.ua"),
        new Student("Lol", "Lolov", "lol@test.com"),
        new Student("Akhtung", "ds", "test@test.com"),
        new Student("Russian", "Schwein", "russian.schwein@gmail.com")
    };

    var repository = new Repository
    {
        Students = students
    };

    Student[] sortedStudents = repository.Sort(new MyComparer());

    var expectedSortedStudents = new List<Student>
    {
        new Student("Akhtung", "ds", "test@test.com"),
        new Student("Lol", "Lolov", "lol@test.com"),
        new Student("Maksym", "Pavliv", "maksym.pavliv.pz.2020@lpnu.ua"),
        new Student("Russian", "Schwein", "russian.schwein@gmail.com")
    };

    Assert.That(sortedStudents, Is.EqualTo(sortedStudents));
}
```

```
[Test]
public void CoursesOrderBy_ByStartDate_OrdersCorrectly()
{
    var repository = new Repository()
    {
        Courses = new List<Course>
        {
            new Course(
                "React course.",
                "Learn React from zero to hero.",
                DateTime.Today.AddDays(1),
                DateTime.Today.AddDays(31)
            ),

            new Course(
                "C# course",
                "Learn C# from zero to hero",
                DateTime.Today.AddDays(8),
                DateTime.Today.AddDays(38)
            ),

            new Course(
                "MAPZ",
                "Become pro with C#!",
                DateTime.Now.AddDays(-60),
                DateTime.Now.AddDays(60)
            )
        }
    };

    var orderedCourses = repository.OrderBy(course => course.StartDate);
    var expectedOrderedCourses = new List<Course>
    {
        new Course(
            "MAPZ",
            "Become pro with C#!",
            DateTime.Now.AddDays(-60),
            DateTime.Now.AddDays(60)
        ),
        new Course(
            "React course.",
            "Learn React from zero to hero.",
            DateTime.Today.AddDays(1),
            DateTime.Today.AddDays(31)
        ),
        new Course(
            "C# course",
            "Learn C# from zero to hero",
            DateTime.Today.AddDays(8),
            DateTime.Today.AddDays(38)
        )
    };

    Assert.That(orderedCourses, Is.EqualTo(expectedOrderedCourses));
}

[Test]
public void EventsOrderBy_ByTitle_OrdersCorrectly()
{
    var repository = new Repository
    {
        Events = new List<Event>
        {
            new Event("Meeting", "Meet and make new friends", new Organizer()),
            new Event("Abrakadabra", "Magic", new Organizer()),
            new Event("Skotoboynia", "Kill russian pigs", new Organizer())
```

```
        }
    };

    var orderedEvents = repository.OrderBy((Event e) => e.Title);
    var expectedOrderedEvents = new List<Event>
    {
        new Event("Abrakadabra", "Magic", new Organizer()),
        new Event("Meeting", "Meet and make new friends", new Organizer()),
        new Event("Skotoboynia", "Kill russian pigs", new Organizer())
    };

    Assert.That(orderedEvents, Is.EqualTo(expectedOrderedEvents));
}

[Test]
public void CoursesGet_AllCourses_ReturnsAllCourses()
{
    var courses = new[]
    {
        new Course(
            "MAPZ",
            "Become pro with C#!",
            DateTime.Now.AddDays(-60),
            DateTime.Now.AddDays(60)
        ),
        new Course(
            "React course.",
            "Learn React from zero to hero.",
            DateTime.Today.AddDays(1),
            DateTime.Today.AddDays(31)
        ),
        new Course(
            "C# course",
            "Learn C# from zero to hero",
            DateTime.Today.AddDays(8),
            DateTime.Today.AddDays(38)
        )
    };

    var repository = new Repository
    {
        Courses = courses
    };

    var actualCourses = repository.GetCourses(_ => true).ToArray();
    Assert.That(actualCourses, Is.EqualTo(courses));

}

[Test]
public void CoursesGet_CoursesTitleStartsWithC_ReturnsCorrectCollection()
{
    var courses = new[]
    {
        new Course(
            "MAPZ",
            "Become pro with C#!",
            DateTime.Now.AddDays(-60),
            DateTime.Now.AddDays(60)
        ),
        new Course(
            "React course.",
            "Learn React from zero to hero.",
            DateTime.Today.AddDays(1),
            DateTime.Today.AddDays(31)
        ),
```

```
            new Course(
                "C# course",
                "Learn C# from zero to hero",
                DateTime.Today.AddDays(8),
                DateTime.Today.AddDays(38)
            )
        };

        var repository = new Repository
        {
            Courses = courses
        };

        var expectedCourses = new[]
        {
            new Course(
                "C# course",
                "Learn C# from zero to hero",
                DateTime.Today.AddDays(8),
                DateTime.Today.AddDays(38)
            )
        };

        var actualCourses = repository.GetCourses(course =>
course.Title.StartsWith("C")).ToArray();
        Assert.That(actualCourses, Is.EqualTo(expectedCourses));
    }
}

public class MyComparer : IComparer<Student>
{
    public int Compare(Student? x, Student? y)
    {
        if (x is null && y is null)
        {
            return 0;
        }

        if (x is null)
        {
            return -1;
        }

        if (y is null)
        {
            return 1;
        }

        return String.Compare(
            x.FullName,
            y.FullName,
            StringComparison.InvariantCultureIgnoreCase
        );
    }
}
```

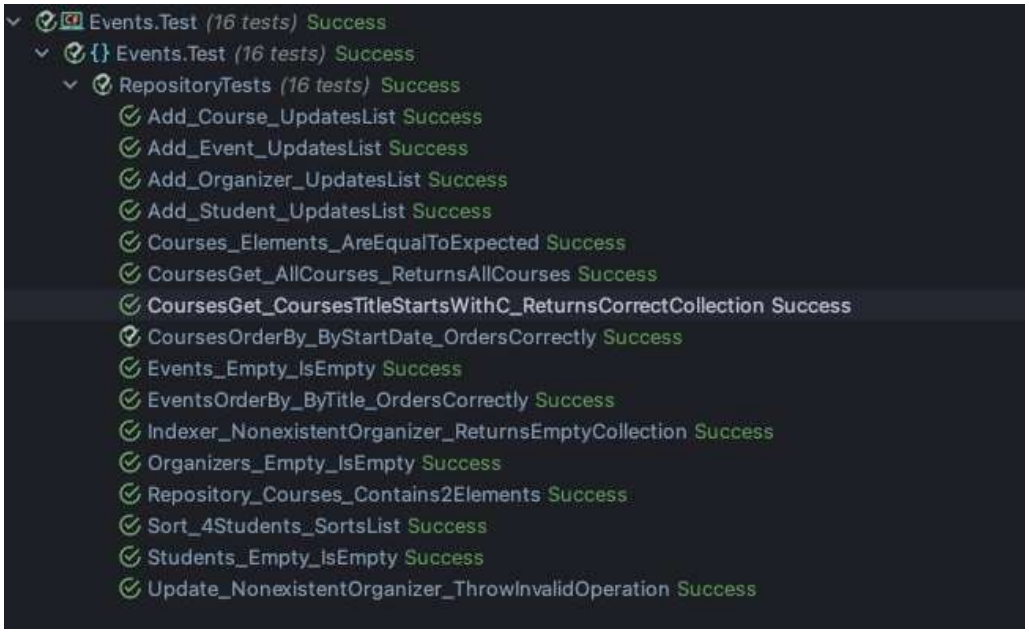

Рис. 1. Всі тести пройшли успішно.

Висновки

На лабораторній роботі я навчився працювати з масивами та структурами List, Dictionary, використовувати технологію LINQ. Написав бібліотеку класів відповідно до мого варіанту проєкту. До написаних класів написав Unit-тести, всі успішно пройшли. В тестах я перевіряв роботу методів, які зокрема використовують LINQ. Показав операції з List, Dictionary, реалізував методи розширювання, показав використання анонімних класів та ініціалізаторів, відсортував список, використовуючи об’єкт інтерфейсу IComparer.