**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

**Звіт**

з комп’ютерного практикуму №2 з дисципліни

«Основи розробки програмного забезпечення на платформі Microsoft.NET»

**«LINQ to XML»**

**Виконав**

(шифр, прізвище, ім'я, по батькові)

*Голяка Д.В.*

**Перевірила**

(прізвище, ім'я, по батькові)

*Ліщук К.І.*

**Варіант 16**

**Умова**

Розробити структуру даних для зберігання інформації про скачки та проведені заїзди. Жокей характеризується псевдонімом, прізвищем, ім’ям, по батькові, датою народження. Коні, які беруть участь в перегонах, характеризуються породою, кличкою і віком. Заїзди проводяться в певну дату і час, мають певний номер (в один день може бути проведено кілька заїздів). Учасники заїзду характеризуються номером і зайнятим місцем в заїзді. Під учасником розуміється жокей і кінь, на якому він буде брати участь в заїзді

**ER-діаграма**

Изображение выглядит как текст, диаграмма, План, Технический чертеж

Автоматически созданное описание

**Опис архітектури**

Проєкт був декомпозований на наступні каталоги для зручності роботи:

DataCollector – представляє контекст даних, клас, в якому агреговано колекції даних;

Entities – представляє набір сутностей (адреси, жокеї, коні та інші)

DTO – представляє об’єкти для отримання результату та зручності передачі.

ER-diagram – представляє ER-діаграму заданого предметного середовища

Execution – представляє набір класів для виконання додатку

Inserters – представляє класи для вставки даних у DataContext

Service – допоміжні класи, що відповідають за логіку роботи програми

FileManagement – представляє класи, що відповідаються за серіалізацію та десеріалізацію.

Extensions – представляє класи розширення

Більш детально із архітектурою додатку можна ознайомитися за посиланням: [GitHub](https://github.com/DmytroHoliaka/labs-dotnet)

**Демонстрація основних функцій**

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

**Запити**

/// <summary>

/// Query 1. Gets all horses of the specified breed

/// </summary>

public IEnumerable<HorseInfo> GetHorsesByBreed(string? breed)

{

if (breed is null)

{

ErrorNotify?.Invoke("Breed cannot be null");

return Enumerable.Empty<HorseInfo>();

}

IEnumerable<HorseInfo>? res = \_xDoc.Root

?.Element("Horses")

?.Elements()

.Where(e => e.Element("Breed")?.Value == breed)

.Select(e => new HorseInfo

{

Id = int.TryParse(e.Attribute("Id")!.Value, out int id)

? id : int.MinValue,

Nickname = e.Element("Nickname")!.Value,

Breed = e.Element("Breed")!.Value,

});

return res ?? Enumerable.Empty<HorseInfo>();

}

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

/// <summary>

/// Query 2. Gets all jockeys from the specified country

/// </summary>

public IEnumerable<JockeyCountry> GetJockeysFromCountry(string? country)

{

if (country is null)

{

ErrorNotify?.Invoke("Country cannot be null");

return Enumerable.Empty<JockeyCountry>();

}

IEnumerable<XElement>? addresses = \_xDoc

.Root!

.Element("Addresses")!

.Elements();

IEnumerable<JockeyCountry>? res = \_xDoc

.Root?.Element("Jockeys")?.Elements()

.Join(addresses ?? Enumerable.Empty<XElement>(),

j => j.Element("AddressId")?.Value,

a => a.Attribute("Id")?.Value,

(j, a) => new JockeyCountry

{

Id = int.TryParse(j.Attribute("Id")?.Value, out int id)

? id : int.MinValue,

FullName = j.Element("FirstName")?.Value + " "

+ j.Element("MiddleName")?.Value + " "

+ j.Element("SecondName")?.Value,

Country = a.Element("Country")?.Value,

})

.Where(p => p.Country == country);

return res ?? Enumerable.Empty<JockeyCountry>();

}

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

/// <summary>

/// Query 3. Gets the stadium with the longest track

/// </summary>

public StadiumInfo GetStadiumWithLongestTrack()

{

StadiumInfo? res = \_xDoc

.Root?.Element("Stadiums")?.Elements()

.Select(se => new StadiumInfo

{

Id = int.TryParse(se.Attribute("Id")?.Value, out int id)

? id : int.MinValue,

Name = se.Element("Name")?.Value,

TrackLength = int.TryParse(se.Element("TrackLength")?.Value, out int length)

? length : int.MinValue,

})

.MaxBy(se => se.TrackLength);

return res ?? new StadiumInfo()

{

Id = int.MinValue,

Name = "Unknown",

TrackLength = int.MinValue

};

}

Изображение выглядит как текст, Шрифт, снимок экрана, Графика

Автоматически созданное описание

/// <summary>

/// Query 4. Gets the jockey who won the most times

/// </summary>

public JockeyWins GetTopWinningJockey()

{

IEnumerable<Participant>? participants = GetParticipants();

IEnumerable<DetailResult>? results = GetDetailResults();

IEnumerable<Jockey>? jockeys = GetJockeys();

JockeyWins? res = jockeys

?.Join(participants ?? Enumerable.Empty<Participant>(),

j => j.Id,

p => p.JockeyId,

(j, p) => new

{

j.Id,

FullName = j.FirstName + " " + j.MiddleName + " " + j.SecondName,

ParticipantId = p.Id,

})

.Join(results ?? Enumerable.Empty<DetailResult>(),

t => t.ParticipantId,

r => r.ParticipantId,

(a, r) => new

{

a.Id,

a.FullName,

r.Place,

})

.GroupBy(g => new { g.Id, g.FullName },

v => v.Place,

(k, group) => new JockeyWins

{

Id = k.Id,

FullName = k.FullName,

WinsCount = group.Count(p => p == 1),

})

.MaxBy(a => a.WinsCount);

return res ?? new JockeyWins

{

Id = int.MinValue,

FullName = "<unknown>",

WinsCount = int.MinValue,

};

}

Изображение выглядит как текст, Шрифт, часы, снимок экрана

Автоматически созданное описание

/// <summary>

/// Query 5. Gets the races that were held in the specified year

/// </summary>

public IEnumerable<RaceInfo> GetRacesByYear(int year)

{

IEnumerable<RaceInfo>? res = \_xDoc

.Root?.Element("Races")?.Elements()

.Select(re => new RaceInfo

{

Id = re.ParseAttributeToInt("Id"),

Number = re.ParseElementToInt("Number"),

Date = re.ParseElementToDateTime("StartDate"),

})

.Where(r => r.Date.Year == year);

return res ?? Enumerable.Empty<RaceInfo>();

}

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

/// <summary>

/// Query 6. Gets the jockeys who participated in the race more than once

/// </summary>

public IEnumerable<JockeyRaceCount> GetJockeysWithMultipleRaces()

{

IEnumerable<JockeyRaceCount> res =

from p in \_xDoc.Root?.Element("Participants")?.Elements() ?? Enumerable.Empty<XElement>()

group p.Attribute("Id")?.Value by p.Element("JockeyId")?.Value into grouped

let raceCount = grouped.Count()

where raceCount > 1

select new JockeyRaceCount

{

Id = grouped.Key.ParseToInt(),

RaceCount = raceCount,

};

return res;

}

Изображение выглядит как текст, Шрифт, снимок экрана, часы

Автоматически созданное описание

/// <summary>

/// Query 7. Gets caretakers who live in the specified city

/// </summary>

public IEnumerable<CaretakerCity> GetCaretakersByCity(string? city)

{

if (city is null)

{

ErrorNotify?.Invoke("City cannot be null");

return Enumerable.Empty<CaretakerCity>();

}

IEnumerable<Address>? addresses = GetAddresses();

IEnumerable<Caretaker>? caretakers = GetCaretakers();

IEnumerable<CaretakerCity>? res = caretakers

?.Join(addresses ?? Enumerable.Empty<Address>(),

c => c.AddressId,

a => a.Id,

(c, a) => new CaretakerCity

{

Id = c.Id,

FullName = c.FirstName + ' ' + c.MiddleName + ' ' + c.SecondName,

City = a.City

})

.Where(t => t.City == city);

return res ?? Enumerable.Empty<CaretakerCity>();

}

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

/// <summary>

/// Query 8. Gets caretakers who care for the specified horse breed

/// </summary>

public IEnumerable<CaretakerBreed> GetCaretakersForBreed(string? breed)

{

if (breed is null)

{

ErrorNotify?.Invoke("Breed cannot be null");

return Enumerable.Empty<CaretakerBreed>();

}

IEnumerable<Horse>? horses = GetHorses();

IEnumerable<Caretaker>? caretakers = GetCaretakers();

IEnumerable<CaretakerBreed>? res = caretakers

?.SelectMany(c => c.HorseIds!,

(c, horseId) => new

{

c.Id,

FullName = c.FirstName + ' ' + c.MiddleName + ' ' + c.SecondName,

HorseId = horseId

})

.Join(horses ?? Enumerable.Empty<Horse>(),

c => c.HorseId,

h => h.Id,

(c, h) => new

{

c.Id,

c.FullName,

h.Breed

})

.Where(t => t.Breed == breed)

.GroupBy(t => new { t.Id, t.FullName, t.Breed },

t => t.Breed,

(k, grouped) => new CaretakerBreed

{

Id = k.Id,

FullName = k.FullName,

Breed = k.Breed,

HorseCount = grouped.Count()

});

return res ?? Enumerable.Empty<CaretakerBreed>();

}

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

/// <summary>

/// Query 9. Gets the horse that has the maximum speed

/// </summary>

public HorseSpeed GetFastestHorse()

{

IEnumerable<Horse>? horses = GetHorses();

IEnumerable<Participant>? participants = GetParticipants();

IEnumerable<DetailResult>? results = GetDetailResults();

HorseSpeed? res = horses

?.Join(participants ?? Enumerable.Empty<Participant>(),

h => h.Id,

p => p.HorseId,

(h, p) => new

{

h.Id,

h.Nickname,

ParticipantId = p.Id,

})

.Join(results ?? Enumerable.Empty<DetailResult>(),

t => t.ParticipantId,

r => r.ParticipantId,

(t, r) => new HorseSpeed

{

Id = t.Id,

Nickname = t.Nickname,

MaxSpeed = r.MaxSpeed

})

.MaxBy(t => t.MaxSpeed);

return res ?? new HorseSpeed

{

Id = int.MinValue,

Nickname = "<unknown>",

MaxSpeed = double.MinValue,

};

}

Изображение выглядит как текст, Шрифт, снимок экрана, часы

Автоматически созданное описание

/// <summary>

/// Query 10. Gets Jockeys older than the specified number of years

/// /// </summary>

public IEnumerable<JockeyAge> GetJockeysByAge(int age)

{

IEnumerable<JockeyAge> res =

from je in \_xDoc.Root?.Element("Jockeys")?.Elements() ?? Enumerable.Empty<XElement>()

let birthDate = je.ParseElementToDateTime("BirthDate")

let currentAge = birthDate != DateTime.MinValue

? Math.Round((DateTime.Now - birthDate).Days / 365.25, 2)

: double.MinValue

where currentAge > age

select new JockeyAge

{

Id = je.ParseAttributeToInt("Id"),

FullName =

je.Element("FirstName")?.Value + " "

+ je.Element("MiddleName")?.Value + " "

+ je.Element("SecondName")?.Value,

Age = currentAge,

};

return res;

}

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

/// <summary>

/// Query 11. Gets Jockeys who have participated in a race on a horse with the specified nickname

/// </summary>

public IEnumerable<JockeyHorseCount> GetJockeysForHorseNickname(string? nickname)

{

if (nickname is null)

{

ErrorNotify?.Invoke("Horse nickname cannot be null");

return Enumerable.Empty<JockeyHorseCount>();

}

IEnumerable<Horse> horses = GetHorses();

IEnumerable<Participant> participants = GetParticipants();

IEnumerable<Jockey> jockeys = GetJockeys();

IEnumerable<JockeyHorseCount> res =

from h in horses

where h.Nickname == nickname

join p in participants on h.Id equals p.HorseId

join j in jockeys on p.JockeyId equals j.Id

let FullName = j.FirstName + ' ' + j.MiddleName + ' ' + j.SecondName

group h.Nickname by new { j.Id, FullName, h.Nickname } into grouped

select new JockeyHorseCount

{

Id = grouped.Key.Id,

FullName = grouped.Key.FullName,

Nickname = grouped.Key.Nickname,

RaceCount = grouped.Count(),

};

return res;

}

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

/// <summary>

/// Query 12. Gets the races where horses older than the specified age participated

/// </summary>

public IEnumerable<RaceHorseCount> GetRacesWithOlderHorses(int age)

{

IEnumerable<Horse> horses = GetHorses();

IEnumerable<Participant> participants = GetParticipants();

IEnumerable<Race> races = GetRaces();

IEnumerable<RaceHorseCount> res =

from h in horses

where h.Age > age

join p in participants on h.Id equals p.HorseId

join r in races on p.RaceId equals r.Id

group h by r into grouped

select new RaceHorseCount

{

Id = grouped.Key.Id,

Number = grouped.Key.Number,

HorseCount = grouped.Count()

};

return res;

}

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

/// <summary>

/// Query 13. Receives caretakers who earn more than the specified amount of money

/// </summary>

public IEnumerable<CaretakerInfo> GetCaretakersByEarnings(int salary)

{

IEnumerable<CaretakerInfo> res =

from ce in \_xDoc.Root?.Element("Caretakers")?.Elements() ?? Enumerable.Empty<XElement>()

let currentSalary = ce.ParseElementToInt("Salary")

where currentSalary > salary

select new CaretakerInfo

{

Id = ce.ParseAttributeToInt("Id"),

FullName =

ce.Element("FirstName")?.Value + " "

+ ce.Element("MiddleName")?.Value + " "

+ ce.Element("SecondName")?.Value,

Salary = currentSalary

};

return res;

}

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

/// <summary>

/// Query 14. Receives horses that have a vet

/// </summary>

public IEnumerable<HorseInfo> GetHorsesWithVet()

{

IEnumerable<HorseInfo> res =

from he in \_xDoc.Root?.Element("Horses")?.Elements() ?? Enumerable.Empty<XElement>()

from i in he.Element("CaretakerIds")?.Elements() ?? Enumerable.Empty<XElement>()

join ce in \_xDoc.Root?.Element("Caretakers")?.Elements() ?? Enumerable.Empty<XElement>()

on i.Value equals ce.Attribute("Id")?.Value

where ce.Element("Responsibility")?.Value == "Veterinary Care"

let Id = he.Attribute("Id")?.Value

let Breed = he.Element("Breed")?.Value

let Nickname = he.Element("Nickname")?.Value

group ce.Attribute("Id")?.Value by new { Id, Breed, Nickname } into grouped

select new HorseInfo

{

Id = grouped.Key.Id.ParseToInt(),

Nickname = grouped.Key.Nickname,

Breed = grouped.Key.Breed,

};

return res;

}

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

/// <summary>

/// Query 15. Gets the stadium where the absolute speed record was set

/// </summary>

public StadiumSpeed GetRecordSpeedStadium()

{

IEnumerable<StadiumSpeed> details =

from se in \_xDoc.Root?.Element("Stadiums")?.Elements() ?? Enumerable.Empty<XElement>()

join re in \_xDoc.Root?.Element("Races")?.Elements() ?? Enumerable.Empty<XElement>()

on se.Attribute("Id")?.Value equals re.Element("StadiumId")?.Value

join pe in \_xDoc.Root?.Element("Participants")?.Elements() ?? Enumerable.Empty<XElement>()

on re.Attribute("Id")?.Value equals pe.Element("RaceId")?.Value

join de in \_xDoc.Root?.Element("DetailResults")?.Elements() ?? Enumerable.Empty<XElement>()

on pe.Attribute("Id")?.Value equals de.Element("ParticipantId")?.Value

select new StadiumSpeed

{

Id = se.ParseAttributeToInt("Id"),

Name = se.GetElementValue("Name"),

Speed = de.ParseElementToDouble("MaxSpeed"),

};

StadiumSpeed? res = details.MaxBy(d => d.Speed);

return res ?? new StadiumSpeed

{

Id = int.MinValue,

Name = "<unknown>",

Speed = double.MinValue,

};

}

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

/// <summary>

/// Query 16. Gets all participants

/// </summary>

public IEnumerable<Participant> GetParticipants()

{

IEnumerable<Participant>? participants = \_xDoc

.Root?.Element("Participants")?.Elements()

.Select(e => new Participant

{

Id = e.ParseAttributeToInt("Id"),

Number = e.ParseElementToInt("Number"),

JockeyId = e.ParseElementToInt("JockeyId"),

HorseId = e.ParseElementToInt("HorseId"),

RaceId = e.ParseElementToInt("RaceId"),

});

return participants ?? Enumerable.Empty<Participant>();

}

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

/// <summary>

/// Query 17. Gets all detail results

/// </summary>

public IEnumerable<DetailResult> GetDetailResults()

{

IEnumerable<DetailResult>? results = \_xDoc

.Root?.Element("DetailResults")?.Elements()

.Select(e => new DetailResult

{

Id = e.ParseAttributeToInt("Id"),

Place = e.ParseElementToInt("Place"),

Duration = e.ParseElementToInt("Duration"),

MaxSpeed = e.ParseElementToDouble("MaxSpeed"),

ParticipantId = e.ParseElementToInt("ParticipantId"),

});

return results ?? Enumerable.Empty<DetailResult>();

}

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

/// <summary>

/// Query 18. Gets all jockeys

/// </summary>

public IEnumerable<Jockey> GetJockeys()

{

IEnumerable<Jockey>? jockeys = \_xDoc

.Root?.Element("Jockeys")?.Elements()

.Select(e => new Jockey

{

Id = e.ParseAttributeToInt("Id"),

FirstName = e.GetElementValue("FirstName"),

MiddleName = e.GetElementValue("MiddleName"),

SecondName = e.GetElementValue("SecondName"),

Pseudonym = e.GetElementValue("Pseudonym"),

BirthDate = e.ParseElementToDateTime("BirthDate"),

AddressId = e.ParseAttributeToInt("AddressId"),

});

return jockeys ?? Enumerable.Empty<Jockey>();

}

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

/// <summary>

/// Query 19. Gets all addresses

/// </summary>

public IEnumerable<Address> GetAddresses()

{

IEnumerable<Address>? addresses = \_xDoc

.Root?.Element("Addresses")?.Elements()

.Select(e => new Address

{

Id = e.ParseAttributeToInt("Id"),

Country = e.GetElementValue("Country"),

City = e.GetElementValue("City"),

Area = e.GetElementValue("Area"),

District = e.GetElementValue("District"),

PostalCode = e.GetElementValue("PostalCode"),

Street = e.GetElementValue("Street"),

BuildingNumber = e.ParseElementToInt("BuildingNumber"),

});

return addresses ?? Enumerable.Empty<Address>();

}

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

/// <summary>

/// Query 20. Gets all caretakers

/// </summary>

public IEnumerable<Caretaker> GetCaretakers()

{

IEnumerable<Caretaker>? caretakers = \_xDoc

.Root?.Element("Caretakers")?.Elements()

.Select(e => new Caretaker

{

Id = e.ParseAttributeToInt("Id"),

FirstName = e.GetElementValue("FirstName"),

SecondName = e.GetElementValue("SecondName"),

MiddleName = e.GetElementValue("MiddleName"),

EmploymentDate = e.ParseElementToDateTime("EmploymentDate"),

Salary = e.ParseElementToInt("Salary"),

Responsibility = e.GetElementValue("Responsibility"),

AddressId = e.ParseElementToInt("AddressId"),

HorseIds = e.ParseElementToList("HorseIds"),

});

return caretakers ?? Enumerable.Empty<Caretaker>();

}

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

/// <summary>

/// Query 21. Gets all horses

/// </summary>

public IEnumerable<Horse> GetHorses()

{

IEnumerable<Horse>? horses = \_xDoc

.Root?.Element("Horses")?.Elements()

.Select(e => new Horse

{

Id = e.ParseAttributeToInt("Id"),

Breed = e.GetElementValue("Breed"),

Nickname = e.GetElementValue("Nickname"),

Age = e.ParseElementToInt("Age"),

CaretakerIds = e.ParseElementToList("CaretakerIds")

});

return horses ?? Enumerable.Empty<Horse>();

}

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

/// <summary>

/// Query 22. Gets all races

/// </summary>

public IEnumerable<Race> GetRaces()

{

IEnumerable<Race>? races = \_xDoc

.Root?.Element("Races")?.Elements()

.Select(e => new Race

{

Id = e.ParseAttributeToInt("Id"),

Number = e.ParseElementToInt("Number"),

StartDate = e.ParseElementToDateTime("StartDate"),

StadiumId = e.ParseElementToInt("StadiumId"),

});

return races ?? Enumerable.Empty<Race>();

}

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

/// <summary>

/// Query 23. Gets all stadiums

/// </summary>

public IEnumerable<Stadium> GetStadiums()

{

IEnumerable<Stadium>? stadiums = \_xDoc

.Root?.Element("Stadiums")?.Elements()

.Select(e => new Stadium

{

Id = e.ParseAttributeToInt("Id"),

Name = e.GetElementValue("Name"),

TrackLength = e.ParseElementToInt("TrackLength"),

ParticipantCapacity = e.ParseElementToInt("ParticipantCapacity"),

AudienceCapacity = e.ParseElementToInt("AudienceCapacity"),

AddressId = e.ParseElementToInt("AddressId"),

});

return stadiums ?? Enumerable.Empty<Stadium>();

}

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

**Висновок**

Упродовж виконання лабораторної роботи був розроблений додаток, який виконує LINQ запити до XML документа. Крім цього на практиці були закріплені такі важливі концепції роботи як серіалізації об’єктів у XML формат та їх десереалізація назад до XML формату. Також був реалізований ряд концепцій, які дозволили глибше зрозуміти мову програмування C#. Я використав методи розширення для роботи із рядками та класом XElement, що дозволило уникнути дублювань та зробити мій код значно лаконічнішим; події для виведення інформації через вказаний делегат, що дозволило мені реалізувати виведення результатів, запитань та помилок із використанням різних кольорів лише за допомогою прив’язви конкретного обробника до екземпляра події.