Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

БАКАЛАВРСЬКА ДИПЛОМНА РОБОТА

на тему:

Підсистема збереження даних обліку виставкового салону з вибірковим відновленням

ПОЯСНЮВАЛЬНА ЗАПИСКА

08-23.БДР.005.00.000 ПЗ

Виконала студентка 2 курсу групи КІ-20мсз
спеціальності 123 Комп'ютерна інженерія
_____ Довгань О.В.
Керівник к.т.н., доц.
_____ Черняк О.І.
"14" 06 2022 р.
Рецензент к.т.н., доц., зав.каф. МБІС
_____ Карпінець В.В.
"15" 06 2022 р.

Допущено до захисту
д.т.н., проф. Азаров О.Д.
"16" 06 2022 р.

ВНТУ 2022

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

Освітньо-кваліфікаційний рівень — бакалавр

Спеціальність — 123 Комп'ютерна інженерія

ЗАТВЕРДЖУЮ

Зав. кафедри ОТ, проф., д.т.н.

_____О.Д. Азаров

"08" 02 2022 р.

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ

студентці **Довгань Олександрі Вячеславівні**

1 Тема роботи «Підсистема збереження даних обліку виставкового салону з вибірковим відновленням» керівник роботи Черняк Олександр Іванович, к.т.н., доц., затверджено наказом вищого навчального закладу від 24 березня 2022 року № 66

2 Строк подання студентом роботи 17 травня 2022 року.

3 Вихідні дані до роботи: технічний опис програмного застосунку, мова програмування C#, віконний додаток, середовище розробки Microsoft Visual Studio.

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): вступ, аналіз сучасного стану технологій програмування, аналіз предметної області, реалізація сховища даних інтерфейсу введення даних, реалізація інтерфейсу введення даних, реалізація класів предметної області, реалізація коду для створення і відображення графа об'єктів.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): схема класів репозиторія, схема ієрархії класів форм, схема класів предметної області.

Консультанти розділів роботи наведені в таблиці 1.

Таблиця 1 — Консультанти роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Черняк О.І., к.т.н., доц.		

7 Дата видачі завдання 14 лютого 2022 року.

8 Календарний план наведено в таблиці 2.

Таблиця 2 — Календарний план

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Постановка задачі роботи	14.02.22	Виконано
2	Аналіз сучасного стану технологій програмування	15.02.22 —15.03.22	Виконано
3	Аналіз предметної області	16.03.22 — 28.03.22	Виконано
4	Реалізація сховища даних інтерфейсу введення даних	29.03.22 — 08.04.22	Виконано
5	Реалізація інтерфейсу введення даних	09.04.22 —29.04.22	Виконано
6	Реалізація коду для створення і відображення графа об’єктів	30.04.22 —27.05.22	Виконано
7	Оформлення пояснювальної записки та ілюстративного матеріалу	28.05.22 — 06.06.22	Виконано
8	Перевірка якості виконання бакалаврської роботи та усунення недоліків	06.06.22	Виконано

Студентка _____ Довгань О.В.

Керівник _____ Черняк О.І.

АНОТАЦІЯ

Тема роботи «Підсистема збереження даних обліку виставкового салону з вибірковим відновленням», автор Довгань Олександра Вячеславівна.

Пояснювальна записка містить 70 сторінок, 16 рисунків, 4 таблиці та 21 лістингів.

Дану бакалаврську дипломну роботу присвячено створенню підсистеми збереження даних програми обліку для виставкового салону.

Реалізація даної роботи була створена так, щоб користувач, який вперше бачить даний програмний продукт, міг швидко і легко освоїти його інтерфейс. Також вона дає можливість користувачеві працювати у режимі системи управління базою даних.

Для забезпечення діалогового спілкування між користувачем і програмою використано: форми — для простої роботи із сховищем даних, функції та їх оператори виклику, інтерфейси вводу/виводу даних, створення графа об’єктів і забезпечення взаємодії цих об’єктів між собою.

Ключові слова: підсистема збереження даних, облік даних, вибіркове відновлення.

ABSTRACT

Theme of the work "Subsystem for saving the accounting data of the exhibition hall with selective restoration", author Dovhan Oleksandra.

The explanatory note contains 47 pages, 15 figures, 4 tables and 14 listings.

This bachelor's thesis is devoted to the creation of a subsystem for storing data accounting programs for the showroom.

The implementation of this project was created so that the user who saw this software product for the first time could quickly and easily master its interface. It also allows users to work in database management mode.

To ensure dialog communication between the user and the program used: forms — for easy work with the data warehouse, functions and their call operators, data input/output interfaces, creating a graph of objects and ensuring the interaction of these objects with each other.

Keywords: data storage subsystem, data accounting, selective recovery.

ЗМІСТ

ВСТУП

Для створення програми використано середовище розробки — Microsoft Visual Studio. Мовою програмування була обрана C#, так як це було однією з умов створення програми.

C# — об'єктно-орієнтована мова програмування,система типізації якої є з безпечною для платформи .NET, розроблена Скотом Вілтамутом, Андерсом Гейлсбергом та Пітером Гольде за підтримки Microsoft Research (при фірмі Microsoft).

Синтаксис C# близький до C++ і Java. Мова має суворо статичну типізацію, підтримує поліморфізм, перевантаження операторів, вказівники на функції — члени класів, події, атрибути, властивості, винятки, коментарі у форматі XML. Дізнавшись багато чого у своїх попередників - C++, Delphi, Module і Smalltalk — C#, виходячи з практики їх використання, виключає деякі моделі, які виявилися очевидними.

Visual Studio Professional — це передове рішення для розробки, що дозволяє командам будь-якого розміру проєктувати і створювати привабливі програми, які зможуть задовольнити, навіть, найвимогливіших замовників. Надає можливість використовувати інструменти гнучкого планування — такі як планування обсягу робіт, панелі завдань і керування невиконаною роботою (для впровадження методів послідовної розробки) і застосування гнучких методологій в зручному для вас темпі. Дозволяє використовувати розширені засоби моделювання, виявлення і проєктування архітектури, щоб описати свою систему і забезпечити повну реалізацію її концепції архітектури. Також є можливість підвищувати якість і скорочувати час усунення неполадок шляхом створення помилок з розгорнутого програмного забезпечення , що включають конкретний перелік дій, і забезпечувати взаємодію з операторами для отримання даних, які дозволять розробникам більш детально аналізувати виробничі проблеми.

Отже, ми маємо можливість створювати високоякісні інноваційні рішення, знижуючи при цьому витрати на розробку .

Windows Forms — це інтерфейс програмного забезпечення (API), який відповідає за графічний інтерфейс користувача і є частинкою Microsoft NET Framework. Цей інтерфейс полегшує доступ до компонентів інтерфейсу Microsoft Windows, створюючи жорсткий диск для вже існуючого Win32 API в керованому коді. Однак керований код - класи, що забезпечують реалізацію API для Windows Forms, не залежно від мови проблеми. Це означає, що розробник все ще може користуватись Windows Forms під час написання програмного забезпечення на C #, C ++, VB.Net, J # тощо.

Створення підсистеми обліку даних дозволить отримати можливість оптимізувати час та робочий процес, що є **актуальною задачею.**

Метою роботи є розробка підсистеми збереження даних обліку виставкового салону з вибірковим відновленням в середовищі Visual Studio 2019, що дозволить структурувати дані та оптимізувати робочий час.

Задачі дослідження бакалаврської роботи:

— проаналізувати предметну область;

— реалізувати сховище даних;

— реалізувати інтерфейс введення даних;

— реалізувати класи предметної області;

— реалізувати код для створення і відображення графа об’єктів;

— реалізувати вибіркове відновлення.

Об’єкт дослідження — процес створення підсистеми збереження даних обліку виставкового салону з вибірковим відновленням.

Предмет дослідження — додаток, що використовується для обліку даних виставкового салону.

Методи дослідження бакалаврської роботи: у роботі використано принципи об’єктно-орієнтованого програмування мовою C# для реалізації запропонованого підходу.

Апробація результатів бакалаврської роботи зроблена в доповіді на Всеукраїнській науково-практичній інтернет-конференції Молодь в науці: дослідження, проблеми, перспективи.

Матеріали роботи доповідались та опубліковувались [1]:

О. І. Черняк О. В. Довгань / підсистема збереження даних обліку виставкового салону з вибірковим відновленням //Тези доповіді. Всеукраїнська науково-практична інтернет-конференція Молодь в науці: дослідження, проблеми, перспективи. Вінниця 2022 р. Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2022/paper/view/14566

Практичне значення отриманих результатів полягає в можливості використання розробленої підсистеми у виставковому салоні для обліку даних.

1 ОГЛЯД СТАНУ ТЕХНОЛОГІЙ ПРОГРАМУВАННЯ

1.1 Об'єктно-орієнтований підхід

Розглянемо особливості об'єктно-орієнтованого підходу до програмування в порівнянні з функціональним підходом. Нагадаємо, що класифікація підходів до програмування була побудована нами у вступній лекції.

Найважливішим кроком на шляху до вдосконалення усіх мов програмування стала поява об'єктно-орієнтованого підходу до програмування (або, скорочено, ООП) та належного класу мов. Саме дослідження теорії та практики проектування і реалізації програмних систем за принципами ООП і є основною метою другої частини даного курсу.

При об'єктно-орієнтованому підході додаткок є описом об'єктів, агрегатів (або класів), їх властивостей (або атрибутів), відносин між ними (способів їх взаємодії) та операцій над об'єктами (або методи)[6].

Безперечною перевагою даного підходу є концептуальна близькість розглянутої предметної області з довільною структурою та цільовим призначенням призначення. Механізм наслідування атрибутів і методів дає змогу будувати похідні поняття на основі базових понять, а отже, створювати модель як будь-який складений предмет із заданими властивостями.

Ще однією теоретично цікавою і практично важливою властивістю об'єктно-орієнтованого підходу є підтримка механізму обробки подій, що змінюють атрибути об'єктів та моделювання їх взаємодії в предметній області.

Якщо переміститись в ієрархії класів від загальних понять заданої предметної області до більш детальних (або від більш складних - до більш простих) і навпаки, програміст отримує можливість змінювати ступінь їх абстрактності або конкретності погляду на реальний світ, що він моделює.

Використання бібліотек об’єктів і методів, що були розроблені раніше (можливо, іншими групами розробників) заощаджує багато роботи в цілому під час виробництва програм, особливо, типового.

Об'єкти, методи і класи, за рахунок поліморфності, можуть зробити більш

гнучким і універсальним реалізоване програмне забезпечення.

Складність відповідної (послідовної та повної) формалізації теорії об’єктів ускладнює перевірку та перевірку розробленого програмного забезпечення. Напевно, ця обставина є одним із найважливіших недоліків об'єктно-орієнтованого підходу до програмування.

Зміна структурно-процедурного підходу на об'єктно-орієнтоване програмування, як і перехід від мов програмування низького рівня до високорівневих, вимагає високорівневого навчання. У той же час ціною цього є підвищення продуктивності програмістів при розробці та впровадженні програмного забезпечення. Ще однією перевагою ООП є більший відсоток повторного застосування вже існуючого програмного коду.

Однак, на відміну від попередніх підходів до програмування, об’єктно-орієнтований підхід вимагає глибокого розуміння основних принципів або, навпаки, концепцій, на яких він базується. Основні концепції ООП зазвичай включають абстракцію даних, успадкування, інкапсуляцію та поліморфізм.

1.2 Основні властивості об'єктів

Інкапсуляція — це механізм, що поєднує дані та методи, обробки цих даних і захищає як від зовнішнього впливу, так і від неправильного використання. Коли методи і данні об’єднуються таким чином — створюється об’єкт [7].

В середині об’єкта дані та методи можуть бути різного ступеня відкритості. Як правило, елементи відкритого класу використовуються для забезпечення інтерфейсу, який керується його закритою частиною.

Сенс інкапсуляції полягає в відділенні реалізації об'єкта (його внутрішнього змісту) від способу взаємодії з ним. Інші об'єкти додатки взаємодіють з даним об'єктом за допомогою наявних у нього відкритих (public) властивостей і методів, які складають його інтерфейс . У загальному вигляді під інтерфейсом розуміється відкритий спосіб взаємодії між різними системами. Якщо інтерфейс класу не буде змінюватися, то додаток зберігає здатність до

взаємодії з його об'єктами, навіть якщо в новій версії класу його реалізація значно зміниться.

Об'єкти можуть взаємодіяти один з одним тільки через свої відкриті методи і властивості, тому об'єкт повинен надавати доступ тільки до тих властивостей і методів, які користувачам необхідні. Інтерфейс ні в якому разі не повинен відкривати доступ до внутрішніх даних об'єкта, тому поля з внутрішніми даними об'єкта зазвичай оголошують з модифікатором private.

Поліморфізм — це властивість, що дозволяє використовувати одне й те саме ім’я для вирішення декількох технічно різних задач, а це означає, що основна мета поліморфізму — використання загальних імен. На практиці це означає, що об’єкти спроможні вибирати внутрішній метод або процедуру, враховуючи тип даних, отриманих в повідомленні [8].

Наприклад, об’єкт з назвою «print» отримує повідомлення, яке виконує дії друку для таких типів даних, як двійкове ціле число, двійкове число з плаваючою комою та символічне число. Залежно від типу даних, які працюють у команді, буде вибрано тип (дані) формату для друку.

Поліморфізму має такі переваги:

— зменшення складності програми;

— можливість використання одного інтерфейсу для одгого класу дій, вибір конкретної дії передається компілятору.

Тобто, поліморфізм означає віднесення певної діяльності до однієї діяльності або позначення, спільне для різних типів об’єктів. Таким чином, кожен об’єкт виконує дії у спосіб, що відповідає його типу.

Наслідування — процес, у якому один об’єкт може набувати властивостей іншого, тобто імітувати властивості іншого об’єкта та додавати характеристики, які є унікальними для нього [7].

Як приклад можна навести класифікацію будь-яких об'єктів, яка відповідає на питання, в чому схожість об'єкта цього класу з іншим об’єктом і в чому різниця. Наслідування забезпечує загальність функції в той час дозволяючи стільки функцій, скільки необхідно. Отже, наслідування забезпечує

в ООП такі функції:

— моделювання концептуальної структури предметної області;

— економія опису, за рахунок використання функції багатократно для завдань різних класів;

— покрокове програмування великих систем, за допомогою багатократної конкретизації класів.

1.3 Visual Studio

Microsoft Visual Studio — серія продуктів фірми Майкрософт, які містять інтегроване середовище розробки програмного забезпечення та низку інших інструментальних засобів [11].

Найбзначиміші версії:

Visual Studio 97 (кодова назва Boston) — це перший випуск Visual Studio, який вперше об’єднує різні інструменти розробки ПЗ. ЇЇ випустили в двох версія Professional і Enterprise і включала Visual Basic 5.0, Visual C++ 5.0, Visual J++ 1.1, Visual FoxPro 5.0, вперше з'явилося середовище розробки ASP — Visual InterDev. Це було першою спробою Microsoft створити єдине середовище розробки для різноманітних мов програмування: Visual C++, Visual J++, Visual InterDev, і MSDN використовували одне середовище, зване Developer Studio. Visual Basic і Visual FoxPro користувалися власними середовищами для розробки.

Visual Studio 6.0 (кодова назва Aspen) — світ побачив дану реалізацію в червні 1998 — остання версія Visual Studio, що для роботи використовує платформу Win9x. Як і раніше користується популярністю серед розробників, які використовували Visual Basic. Ця версія змогла стати основним середовищем розробки додатків під Windows від Microsoft, до появи платформи .NET. Ця версія була основою для розробників Microsoft протягом наступних чотирьох років. Visual Studio 6.0 стала останньою версією, що містила в собі COM версію Visual Basic’a. Це була заключна версія, що містила мову програмування Visual J++. Існували дві версії Visual Studio 6.0: Professional та Enterprise. Enterprise включав в себе додаткові плагіни, які не містились в Professional, включно з

Application Performance Explorer, Automation Manager, Microsoft Visual Modeler, RemAuto Connection Manager, Visual Studio Analyzer.

Visual Studio .NET 2002 (кодова назва Rainier; внутрішня версія 7.0) — вийшла в світ в лютому 2002 (містила .NET Framework 1.0). Service Pack 1 для Visual Studio .NET (2002) випущений в березні 2005. Бета – версією можна було користуватись вже в 2001 році. Маштабною зміною було впровадження менеджера коду. Додатки, які були створені за участі Visual Studio .NET, не компілювались в машинну мову, а записувались у формат, який мав назву Microsoft Intermediate Language (MSIL) або Common Intermediate Language (CIL). Коли MSIL-додаток використовували, він автоматично компілювався в машинну мову для цієї платформи, завдяки цьому код ставав кросплатформним, що створювало можливість виконувати його на різноманітних платформах. Проте такі застосунки могли виконуватись тільки на тих платформах, які підтримували Common Language Infrastructure. Тому користуватись цими додатками в операційних системах Linux або Mac OS можливо було лише з використанням спеціальних програми як, Mono та DotGNU. Пакет був оприлюднений одразу в чотирьох версіях: Professional, Enterprise Developer, Academic та Enterprise Architect. Вперше було представлено нову мову програмування C# (сі–шарп), яка була спеціально створена для застосування в Visual Studio .NET. Також було представлено наслідника Visual J++ що мав назву Visual J#. За участі Visual Studio .NET можна було розробляти звичайні застосунки та вебсайти (використовуючи ASP.NET та Web сервіси). У травні 2005 року випустили пакет оновлень для Visual Studio .NET [12].

Visual Studio .NET 2003 (кодова назва Everett; внутрішня версія 7.1) — оприлюднена в квітні 2003 (містить .NET Framework 1.1). Це виявилась перша версія, що допускала створення додатків для мобільних пристроїв, використовуючи ASP.NET або .NET Compact Framework. Внутрішній номер версії Visual Studio .NET 2003 був 7.1, але файли мали іншу версію, а саме 8.0. Visual Studio .NET 2003 також було оприлюднено в чотирьох варіантах: Academic, Professional, Enterprise Developer, та Enterprise Architect. Версія

Enterprise Architect містила специфічний додаток Microsoft Visio 2002, що давав змогу будувати UML об’єкти. Пакет оновлень для Visual Studio .NET 2003 було оприлюднений 13 вересня 2006 року.

Перший нестабільний випуск наступної версії програми під умовною назвою «15» побачив світ 30 березня 2016 року. Головними змінами стали інтерфейс інсталятора та незліченні незначні покращення у різноманітних компонентах середовища розробки. Очікувана підтримувана мови програмування Solidity.

Версія була оприлюднена під назвою Visual Studio 2017 7 березня 2017 року.

Перша версія нової Visual Studio 19 (Preview) побачила світ у грудні 2018 року. Вона містила 5 етапів. Березень визначив RC (Release Candidate). Тому через місяць, а саме 2 квітня була оприлюднена стабільна версія Visual Studio 2019 на Windows і Visual Studio on MAC. Як і у попередніх версіях є три типи (Professional, Community, Enterprise).

2 АНАЛІЗ СУЧАСНОГО СТАНУ ТЕХНОЛОГІЙ ПРОГРАМУВАННЯ

2.1 C#

C# (вимовляється Сі-шарп) — це об'єктно-орієнтована мова програмування із системою типізації, що безпечна для платформи .NET. Розроблена Андерсом Гейлсбергом, Скотом Вілтамутом та Пітером Гольде під егідою Microsoft Research (належить Microsoft).

Синтаксис C# близький до C++ і Java. Мова має строгу статичну типізацію, підтримує поліморфізм, перевантаження операторів, вказівники на функції-члени класів, атрибути, події, властивості, винятки, коментарі у форматі XML. Перейнявши багато від своїх попередників — мов C++, Object Pascal, Модула і Smalltalk — C#, спираючись на практику їхнього використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем, наприклад, мова C#, на відміну від C++, не передбачає множинне успадкування класів.

Java здобула велику популярність, і була ліцензована компанією Microsoft. Але з часом Sun почала звинувачувати Microsoft у тому, що вона перемогла Java, яка сумісна лише з платформою Windows, у створенні власного клону. Microsoft відмовилася виконувати вимоги Sun, і тому з'ясування стосунків набуло статусу судового процесу. Суд визнав позицію Sun справедливою, і зобов'язав Microsoft утримуватися від неліцензійного використання Java.

У цій ситуації в Microsoft вирішили, користуючись своїм панівним становищем на ринку, створити свій власний аналог Java — мову, для якої корпорація буде повноцінним власником. Ця новостворена мова отримала назву C#. Вона успадкувала від Java концепції віртуальної машини (середовище .NET), байт-коду (MSIL) і більшої безпеки вихідного коду програм, плюс врахувала досвід використання програм на Java [9].

Нововведенням C# стала можливість простішого, порівняно з попередниками, взаємодії з кодом програм, написаних іншими мовами, що важливо при створенні великих проектів. Якщо на платформі .NET запускаються

програми різними мовами, .NET подбає про сумісність програм (тобто типів даних, згідно з остаточним рахунком).

На сьогодні C# визначено як флагманська мова Microsoft, оскільки вона найповніше використовує нові можливості платформи .NET. Решта мов програмування, хоча й підтримуються, але визнані такими, що мають спадкові недоліки щодо використання .NET. Рядок в C# є типом посилання.

У той час як визначення мов C# і CLI стандартизовані ISO та Ecma, що забезпечує розумний і недискримінаційний ліцензійний захист (RAND) від патентних позовів, Microsoft використовує C# і CLI у своїй бібліотеці Base Class Library (BCL), яка є базою їх власної платформи .NET framework, і яка надає низку нестандартизованих класів (розширений I/O, GUI Windows Forms, веб-служби тощо) [10].

У деяких випадках, коли патенти Microsoft пов’язані зі стандартами, що використані у .NET framework, задокументованими Microsoft, то застосовані патенти доступні через інші RAND умови або через Обітницю Відкритої Специфікації Microsoft (Microsoft's Open Specification Promise, OSP), які випускають патентні права публічно [10]. Але є деякі застереження та дискусії про те, щодо того, що є додаткові аспекти, запатентовані Microsoft, які не охоплені, які можуть утримувати незалежних реалізаторів повного фреймворку.

Microsoft також погодилася не судитися з розробниками відкритого програмного забезпечення про порушення прав у неприбуткових проєктах для частини свого фреймворку, покритого OSP [6]. Microsoft погодився не порушувати вимоги до патентів на продукти Novell проти платних клієнтів Novell за винятком списку продуктів, у яких явно не згадуються C#, .NET чи реалізацію .NET від Novell (проєкт Mono). Проте Novell дотримується точки зору, що Mono не порушує жодного патенту Microsoft. Microsoft також уклав спеціальну угоду не позиватися проти браузерного плагіну Moonlight, який спирається на Mono, отриманного від Novell.

У зауваженні, що було опубліковано на сайті новин Free Software Foundation у червні 2009 Річард Столлман застеріг, що на його думку, що

«Microsoft можливо планує одного дня оголосити всі вільні реалізації C# такими, що використовують програмні патенти» і дав рекомендацію розробникам уникати ситуацій, які він називає «безвідплатним ризиком», що можуть бути наслідками «залежністі вільних реалізацій C#». Пізніше Free Software Foundation повторила свої застереження, стверджуючи, що розширення Microsoft Community Promise на специфікації ECMA C# і CLI можуть не захистити від зловживання Microsoft відкритим реалізаціям C#, оскільки багато специфічних для Windows бібліотек, включених у .NET та Mono, не покриті цими обіцянками. Тому більшість провідних дистрибутивів Лінукс, за винятком Novell SUSE Linux, не включають Mono в установку за умовчанням (хоча він доступний до завантаження з репозиторіїв).

2.2 Типи та змінні

У C# існують два різновиди типів: типи посилань і типи значень. Перемінні типу значень містять безпосередньо дані, а змінних посилальних типів зберігаються посилання необхідні дані, що називаються об'єктами. Дві перемінні типу посилання можуть посилатися на той самий об'єкт, тому може статися так, що операції над однією змінною вплинуть на об'єкт, на який посилається інша перемінна. Кожна перемінна типу значення має особисту копію даних, і операції над однією змінною що неспроможні торкатися іншу (крім змінних параметрів ref і out).

Тип визначає структуру та поведінку будь-яких даних у C#. Оголошення типу може включати його члени, базовий тип, інтерфейси, що він реалізує, та операції, дозволені цього типу. Змінна - це мітка, яка посилається на екземпляр певного типу.

Ідентифікатор — це ім'я змінної. Ідентифікатор — це послідовність символів Юнікод без пробілів. Ідентифікатор може бути зарезервованим словом C#, якщо має префікс @. Під час взаємодії з іншими мовами як ідентифікатор можна використовувати зарезервоване слово.

Типи значень C# діляться на прості типи, типи перерахувань, типи структур, типи, що допускають значення NULL, і типи значень кортежів. Посилальні типи C# ділять на типи класів, типи інтерфейсів, типи масивів і типи делегатів.

Далі представлені загальні відомості про систему типів C#.

Типи значень:

— прості типи (ціле зі знаком: short, int, long; ціле без знаку: ushort, uint, ulong; символи Юнікоду; двійкова з плаваючою комою IEEE: double; десяткова точність з плаваючою комою з високою точністю; логічний: bool, що використовується для представлення логічних значень, які можуть мати значення true або false);

— типи перерахування — типи користувача у форматі enum E {...}. Тип enum є окремим типом зі списком іменованих констант. Кожен тип enum має базовий тип, у ролі якого виступає один з восьми цілих типів. Набір значень типу enum аналогічний до набору значень його базового типу;

— типи структур — типи користувача у форматі struct S {...};

— типи значень, що припускають значення NULL — розширення інших типів значень, що припускають значення null;

— типи значень кортежів — типи користувача у форматі (T1, T2, ...).

Посилальні типи:

— типи класів (вихідний базовий клас для всіх типів: object; рядки в Юнікоді — послідовність одиниць коду UTF-16; типи користувача у форматі class C {...});

— типи інтерфейсів — типи користувача у форматі interface I {...};

— типи масивів — одновимірні, багатовимірні масиви та масиви масивів (наприклад, int[], int[,] та int[][]);

— типи делегатів — типи користувача у форматі delegate int D(...).

Програми, написані мовою C#, використовують оголошення типів, щоб створити нові типи. У оголошенні типу потрібно вказати ім'я та члени нового типу. Користувач визначає шість категорій типів C#: типи класів, типи

перерахування, типи структур, типи інтерфейсів, типи делегатів і типи значень кортежів. Можна також оголошувати типи record, або record struct, або record class. Типи записів мають члени, синтезовані компілятором. Записи використовуються в основному для зберігання значень із мінімальною пов'язаною поведінкою.

Тип class визначає структуру даних, що містить дані-члени (поля) та функції-члени (методи, властивості тощо). Механізми одиночного успадкування та поліморфізму, що дають можливість створювати похідні класи, які розширюють та уточнюють визначення базових класів підтримуються класами.

Тип struct схожий на тип класу тим, що він є структурою з даними-членами і функціями-членами. Але, на відміну класів, структури є типами значень і зазвичай вимагають виділення пам'яті з купи. Типи структури не підтримують успадкування, що визначається користувачем, і всі типи структури неявно успадковують від типу object.

Тип interface визначає договір як іменованого набору відкритих елементів. Об'єкт типу class або struct, що реалізує interface, повинен надати реалізацію для всіх елементів інтерфейсу. Тип interface може успадковувати від кількох базових інтерфейсів, а class чи struct можуть реалізовувати кілька інтерфейсів.

Тип delegate (делегат), що являє собою посилання на методи із зазначеним списком параметрів і типом значення, що повертається. Завдяки делегатам є можливість використовувати методи як сутності, при цьому зберігаючи їх у змінній та передаючи як параметри. Делегати подібні до типів функцій, що використовуються у функціональних мовах. Їх принцип роботи схожий до покажчиків функцій деяких мов. На відміну від покажчиків функцій, делегати є об'єктно-орієнтованими та типобезпечними.

Типи class, struct, interface та delegate підтримують універсальні шаблони, які дозволяють передавати їм інші типи як параметри.

C# підтримує одновимірні та багатовимірні масиви будь-якого типу. На відміну від перерахованих вище типів, типи мас сивів не потрібно оголошувати перед використанням. Типи масивів можна сформувати, просто ввівши після

імені типу квадратні дужки. Наприклад, int[] є одномірним масивом значень типу int, а int[,] — двомірним масивом значень типу int, тоді як int[][] є одномірним масивом одномірних масивів (або масивом масивів) значень типу int.

Типи, що допускають NULL, не вимагають окремого визначення. Для кожного звичайного типу T, який не допускає значення NULL, існує ідентичний тип T?, який відрізняється тим, що може містити додаткове значення null. Наприклад, int? є типом, який може містити будь-яке 32-розрядне ціле число або null, а string? - будь-яке значення string чи null.

Система типів C# уніфікована таким чином, що значення будь-якого типу можна розглядати як object (об'єкт). Кожен тип C# є прямо або опосередковано похідним від типу класу object, і цей тип object є вихідним базовим класом для всіх типів. Щоб значення посилання типу оброблялися як об'єкти, їм просто надається тип object. Щоб значення типів значень оброблялися як об'єкти, виконуються операції упаковки-перетворення та розпакування-перетворення. У наступному прикладі значення int перетворюється на object, а потім назад на int.

```
int i = 123;
object o = i; // Boxing
int j = (int) o; // Unboxing
```

Якщо значення типу призначається до посилання object, для зберігання значення виділяється упаковка. Ця упаковка є екземпляром типу посилання, і в неї копіюється значення. І навпаки, якщо посилання типу object використовується для типу значення, для відповідного object виконується перевірка, чи є упаковкою правильного типу. Якщо ця перевірка успішно завершується, копіюється значення цієї упаковки.

Уніфікована система типів C# забезпечує обробку типів значень як object посилання "за запитом". Завдяки уніфікації бібліотеки загального призначення, які використовують тип object, можуть користовуватися усіма типами, похідними від object , включаючи типи посилань і типи значень.

У C# існує кілька типів змінних, у тому числі поля, елементи масиву, локальні перемінні та параметри. Перемінні є місця зберігання, і кожна

перемінна має тип, який визначає допустимі значення для зберігання в цій змінній.

Приклади:

— тип значення, що не допускає значення Null (значення такого типу);

— тип значення, що припускає значення Null (значення null чи значення такого типу);

— object (посилання null, посилання на об'єкт будь-якого посилання або посилання на упаковане значення будь-якого типу значення);

— тип класу (посилання null, посилання на екземпляр такого типу класу або посилання на екземпляр будь-якого класу, похідного від такого типу класу);

— тип інтерфейсу (посилання null, посилання на екземпляр типу класу, який реалізує такий тип інтерфейсу, або посилання на упаковане значення типу значення, яке реалізує такий тип інтерфейсу);

— тип масиву (посилання null, посилання на екземпляр такого типу масиву або посилання на екземпляр будь-якого сумісного типу масиву);

— тип делегата (посилання null або посилання на екземпляр сумісного типу делегата).

2.3 Структура програми

Основними поняттями Організації в C # є програми, простору імен, типи, члени складання. Програма оголошує типи учасників. Ці типи, зазвичай, організовують у просторі імен. Прикладами типів є класи, структури та інтерфейси. Членами є поля, методи, властивості та події. Під час компіляції програми на C# упаковуються у асемблер. Складання зазвичай мають розширення .exe файлу або .dll, залежно від того, чи реалізують вони .exe або бібліотеки відповідно.

Приклад наведено у лістингу 2.1.

Лістинг 2.1 — Приклад структури програми

```
namespace Acme.Collections;
```

```
public class Stack<T>
{
    Entry _top;
    public void Push(T data)
    {
        _top = new Entry(_top, data);
    }
    public T Pop()
    {
        if (_top == null)
        {
            throw new InvalidOperationException();
        }
        T result = _top.Data;
        _top = _top.Next;
        return result;
    }
    class Entry
    {
        public Entry Next { get; set; }
        public T Data { get; set; }
        public Entry(Entry next, T data)
        {
            Next = next;
            Data = data;
        }
    }
}
```

Повне ім'я для цього класу: Acme.Collections.Stack. Цей клас містить кілька членів: поле з ім'ям _top, два методи з іменами Push та Pop, а також вкладений

клас з ім'ям Entry. Клас Entry, у свою чергу, містить три члени: властивість з ім'ям Next, властивість з ім'ям Data та конструктор. Stack Є Stack класом. Він має параметр одного типу T, який замінюється конкретним типом під час використання.

Стек – це колекція типу FILO (прибув першим – обслужений останнім). Нові елементи додаються до стека. Видалений елемент виключається з верхньої частини стека. У попередньому прикладі оголошується тип Stack, який визначає сховище та поведінку для стека. Можна оголосити змінну, яка посилається на екземпляр типу Stack для використання цієї можливості.

Збірки містять код, виконання якого виглядає як інструкції проміжної мови (IL) і символьна інформація у вигляді метаданих. Перед виконанням JIT-компілятор середовища CLR .NET перетворює код IL у складання на код, що залежить від процесора.

Складання повністю описує сама себе і містить весь код і метадані, тому C# не використовуються директиви #include і файли заголовків. Щоб використовувати у програмі C# відкриті типи та члени, які містяться у певній збірці, вам достатньо вказати посилання на цю збірку під час компіляції програми. Зразок використання класу Acme.Collections.Stack зі збирання acme.dll у лістингу 2.2.

Лістинг 2.2 — Зразок використання класу Acme.Collections.Stack

```
class Example
{
   public static void Main()
   {
      var s = new Acme.Collections.Stack<int>();
      s.Push(1); // stack contains 1
      s.Push(10); // stack contains 1, 10
      s.Push(100); // stack contains 1, 10, 100
      Console.WriteLine(s.Pop()); // stack contains 1, 10
      Console.WriteLine(s.Pop()); // stack contains 1
```

```
            Console.WriteLine(s.Pop()); // stack is empty
        }
    }
```

Для компіляції програми потрібно створити посилання на складання, що містить клас стека, визначений у лістингу 2.2.

Програми C# можна зберігати в кількох вихідних файлах. Під час компіляції програми на C# всі вихідні файли обробляються одночасно, тому вони можуть вільно посилатися друн на друга. По суті, це схоже на злиття всіх вихідних файлів в один великий файл перед обробкою. У C# не користуються випереджаючими оголошеннями, так як порядок оголошення, за рідкісним винятком, не грає жодної ролі. C# не вимагає від вас оголошувати лише один відкритий тип в одному вихідному файлі, а ім'я вихідного файлу не може збігатися з типом, оголошеним у цьому файлі.

C# є наближеним родичем мови програмування Java. Мова Java була винайдена компанією Sun Microsystems, коли глобальний поштовх розвитоку інтернету поставив завдання розподілених обчислень. На основі популярної мови C++, Java прибрала з неї потенційно небезпечні речі (наприклад, вказівники без контролю виходу за межі). Для розподілених обчислень була створена концепція віртуальної машини та машинно-незалежного байт-коду як його проміжної бази між вихідним текстом програми та апаратним забезпеченням та навчальними засобами.

Стандартні блоки програм C# :

— члени;

— поля;

— методи;

— інші функції-члени;

— вирази;

— оператори.

2.4 Стандартні блоки програми

Члени класу class є class або членами екземпляра. Статичні учасники належать класу загалом, а учасники екземпляра належать конкретним об'єктам (примірникам класів).

Види компонентів, які можуть бути у класі:

— константи — константні значення пов'язані з класом;

— поля — змінні, пов'язані із класом;

— методи — дії, що можуть виконуватися класом;

— властивості — події, пов'язані з читанням та записом іменованих якостей класу;

— індексатори — дії, що реалізують індексування екземплярів класу, щоб звертатися до них як до масиву;

— події — повідомлення, які можна створити цим класом;

— оператори — підтримувані класом оператори перетворення та вираження;

— конструктори — дії, необхідні для ініціалізації екземплярів класу чи класу загалом;

— методи завершення — дії, які виконуються перед остаточним видаленням екземплярів класу;

— типи — вкладені типи, оголошені класі.

Кожен член класу має певний рівень доступності. Він визначає, з якої галузі програми можна звертатися до цього члена. Існує шість рівнів доступності.

Модифікатори доступу:

— public — доступ не обмежений;

— private — доступ можливий лише з цього класу;

— protected — доступ можливий із цього класу та з класів, похідних від нього;

— internal — доступ обмежений поточною збіркою (.exe або .dll);

— protected internal — доступ обмежений даним класом, класами, похідними від даного класу, або класами у тій самій збірці;

— private protected — доступ обмежений даним класом чи класами, похідними від цього типу у тій самій збірці.

Поле є змінною, пов'язаною з певним класом або екземпляром класу.

Поле, оголошене модифікатором static, є статичним. Статичне поле визначає одне місце зберігання. Незалежно від того, скільки буде створено екземплярів цього класу, існує лише одна копія статичного поля.

Поле, оголошене без static, є полем екземпляра. Кожен екземпляр класу містить окремі копії всіх полів екземпляра, визначених для цього класу.

У лістингу 2.3 кожен екземпляр класу Color містить окрему копію полів екземпляра R, G і B, але для кожного зі статичних полів Black, White, Red, Green та Blue існує лише одна копія.

Лістинг 2.3 — Зразок коду

```
public class Color
{
   public static readonly Color Black = new(0, 0, 0);
   public static readonly Color White = new(255, 255, 255);
   public static readonly Color Red = new(255, 0, 0);
   public static readonly Color Green = new(0, 255, 0);
   public static readonly Color Blue = new(0, 0, 255);
   public byte R;
   public byte G;
   public byte B;
   public Color(byte r, byte g, byte b)
   {
      R = r;
      G = g;
      B = b;
```

```
        }
}
```

Як показано в лістингу 2.3, поля для читання можуть бути оголошені з модифікатором. Привласнення полю, доступному лише читання, може бути виконано лише як частина оголошення поля чи конструкторі у тому класі.

Метод — це член, який реалізує обчислення чи дію, які можуть виконувати об'єкт чи клас. Доступ до статичними методами здійснюється через клас. Доступ до методів екземпляра здійснюється через екземпляр класу.

Для методу можна визначити список параметрів, які представляють значення, що передаються методу, або посилання на змінні. Методи мають тип, що повертається, який задає тип значення, що обчислюється і повертається методом. Тип методу, що повертається -, void якщо він не повертає значення.

Як і типи методи можуть мати набір параметрів типу, для яких при виклику методу необхідно вказувати аргументи типу. На відміну від типів, аргументи типу часто можуть виводитися з аргументів виклику методу, і тоді їх не обов'язково ставити явно.

Сигнатура методу має бути унікальною в межах класу, в якому оголошено цей метод. Сигнатура методу включає ім'я методу, кількість параметрів типу, а також кількість, модифікатори та типи параметрів методу. Сигнатура методу не включає тип значення, що повертається.

Якщо тіло методу є поодиноким виразом, метод можна визначити за допомогою компактного формату виразу:

```
public override string ToString() => "This is an object".
```

Параметри дозволяють передати метод значення або посилання на змінні. Фактичні значення параметрам методу присвоюються з урахуванням аргументів, заданих під час виклику методу. Існує чотири типи параметрів: параметри значення, параметри посилання, параметри виведення та масиви параметрів.

Параметр значення використовується передачі вхідних аргументів. Параметр значення зіставляється з локальною змінною, яка отримає початкове

значення значення аргументу, переданого в цьому параметрі. Зміни параметра значення не впливають на аргумент, надісланий для цього параметра.

Параметри значення можна зробити необов'язковими, вказавши для них значення за промовчанням. Тоді відповідних аргументів можна не вказувати.

Посилання використовується для передачі аргументів за посиланням. Аргумент, що передається до посилального параметра, повинен бути змінною з певним значенням. При виконанні методу посилання параметр вказує на те ж місце зберігання, де розміщена змінна аргументу. Щоб оголосити посилання, використовуйте модифікатор ref. Наступний приклад коду показує використання параметрів ref.

2.5 Основні можливості мови C#

В C# і .NET є безліч різних типів колекцій. Синтаксис масивів визначається язиком. Універсальні типи колекцій перераховані у просторі імен System.Collections.Generic. До спеціалізованих колекцій відносяться System.Span<T> для доступу до безперервної пам'яті в кадрі стека та System.Memory<T> для доступу до безперервної пам'яті в купі. Усі колекції, включаючи масиви, Span<T> та Memory<T>, використовують загальний принцип ітерації.

Використовується інтерфейс System.Collections.Generic.IEnumerable<T>. Цей єдиний принцип означає, що будь-який тип колекцій можна використовувати з запитами LINQ або іншими алгоритмами. Методи пишуться за допомогою IEnumerable<T> і алгоритми працюють з будь-якою колекцією.

Масив — це структура даних, що містить ряд змінних, до яких здійснюється доступ за допомогою обчислюваних індексів. Всі перемінні, що містяться в масиві, також звані елементами масиву, відносяться до одного типу. Цей тип називається типом елемента масиву.

Самі масиви мають тип посилання, і оголошення змінної масиву тільки виділяє пам'ять для посилання на екземпляр масиву. Фактичні екземпляри масиву створюються динамічно під час виконання оператором new. Операція

new задає довжину нового екземпляра масиву, який потім фіксується на час існування екземпляра. Елементи масиву мають індекси в діапазоні від 0 до Length — 1. Оператор new автоматично ініціалізує всі елементи масиву значенням за замовчуванням. Наприклад, всім числових типів встановлюється нульове значення, а всіх посилальних типів — значення null.

Інтерполяція рядків C# дозволяє форматувати рядки шляхом визначення виразів, результати яких розміщуються в рядок формату.

Мова C# надає вирази зіставлення шаблонів для запиту стану об'єкта та виконання коду на основі цього стану. Щоб визначити, яку дію слід зробити, можна перевірити типи та значення властивостей та полів. Також ви можете перевірити елементи списку або масиву. Вираз switch є основним для зіставлення шаблонів.

Тип делегата представляє посилання на методи з певним списком параметрів і типом значення, що повертається. Делегати дозволяють використовувати методи як сутності, зберігаючи їх у змінні та передаючи як параметри. Принцип роботи делегатів близький до вказівників функцій деяких мов. На відміну від покажчиків функцій, делегати є об'єктно-орієнтованими та типобезпечними.

Приклад коду оголошення та використання типу делегата з ім'ям Function представлено у лістингу 2.4.

Лістинг 2.4

```
delegate double Function(double x);

class Multiplier
{
   double _factor;
   public Multiplier(double factor) => _factor = factor;
   public double Multiply(double x) => x * _factor;
}
class DelegateExample
```

```
{
    static double[] Apply(double[] a, Function f)
    {
        var result = new double[a.Length];
        for (int i = 0; i < a.Length; i++) result[i] = f(a[i]);
        return result;
    }
    public static void Main()
    {
        double[] a = { 0.0, 0.5, 1.0 };
        double[] squares = Apply(a, (x) => x * x);
        double[] sines = Apply(a, Math.Sin);
        Multiplier m = new(2.0);
        double[] doubles = Apply(a, m.Multiply);
    }
}
```

Примірник Function з типом делегата може посилатися будь-який метод, який приймає аргумент double і повертає значення double. Метод Apply застосовує заданий Function до елементів double[] та повертає double[] з результатами. У методі Main використовується Apply для застосування трьох різних функцій до double [].

Делегат може посилатися на статичний метод (наприклад, Square або Math.Sin у попередньому прикладі) або метод екземпляра (наприклад, m.Multiply у попередньому прикладі). Делегат, який посилається на метод екземпляра, містить посилання на конкретний об'єкт. Коли метод екземпляра викликається через делегат, цей об'єкт перетворюється на це у виклику.

Делегати можуть створюватися з використанням анонімних функцій або лямбда-виражень, тобто "вбудованих методів", що створюються при оголошенні. Анонімні функції можуть використовувати локальні змінні сусідніх методів.

Делегат немає інформації чи обмежень щодо того, якого класу належить метод, який він посилається. Метод, на який вказує посилання, повинен мати ті ж параметри і тип значення, що повертається, що і делегат.

Типи, члени та інші сутності у програмі C# підтримують модифікатори, які керують деякими аспектами їхньої поведінки. Наприклад, доступність методу визначається за допомогою модифікаторів public, protected, internal та private. C# узагальнює цю можливість, дозволяючи користувачам визначати власні типи декларативних відомостей, призначати їх для сутності програми та вилучати під час виконання. Програми задають ці декларативні відомості шляхом визначення та використання атрибутів.

Усі класи атрибутів є похідними від базового класу Attribute, який надається у бібліотеці .NET. Щоб задати атрибут, його ім'я та можливі аргументи вказуються у квадратних дужках безпосередньо перед оголошенням відповідної сутності. Якщо ім'я атрибута закінчується Attribute, ця частина імені може бути опущена за посиланням на атрибут.

Метадані, визначені атрибутами, можна зчитувати та використовувати під час виконання за допомогою відображення. Коли за допомогою цього методу виконується запит конкретного атрибута, викликається конструктор класу атрибута із зазначенням відомостей, представлених у вихідному коді програми, а потім повертається створений екземпляр атрибута. Якщо додаткові відомості надаються через властивості, перед поверненням екземпляра атрибуту цим властивостям надаються зазначені значення.

3 РОЗРОБКА ПІДСИСТЕМИ ЗБЕРЕЖЕННЯ ДАНИХ

Головні етапи виконання практичного завдання:

— аналіз предметної області (джерела інформації, перелік сутностей, атрибутів та зв’язків);

— реалізація сховища даних (схема класів сховища (ієрархія), схема XML у вигляді таблиць або схема бази даних);

— реалізація інтерфейсу введення даних (опис специфічних форм, схема класів форм (ієрархія));

— реалізація класів предметної області (схема класів ПО (ієрархія));

— реалізація коду для створення і відображення графа об’єктів (схема зв’язків об’єктів класів (асоціація, агрегація, композиція), текстове представлення графа об’єктів).

3.1 Аналіз предметної області

Проаналізувавши свою предметну область — подивившись зразки баз даних було створено базу даних в програмному середовищі Microsoft Access (рис. 3.1).

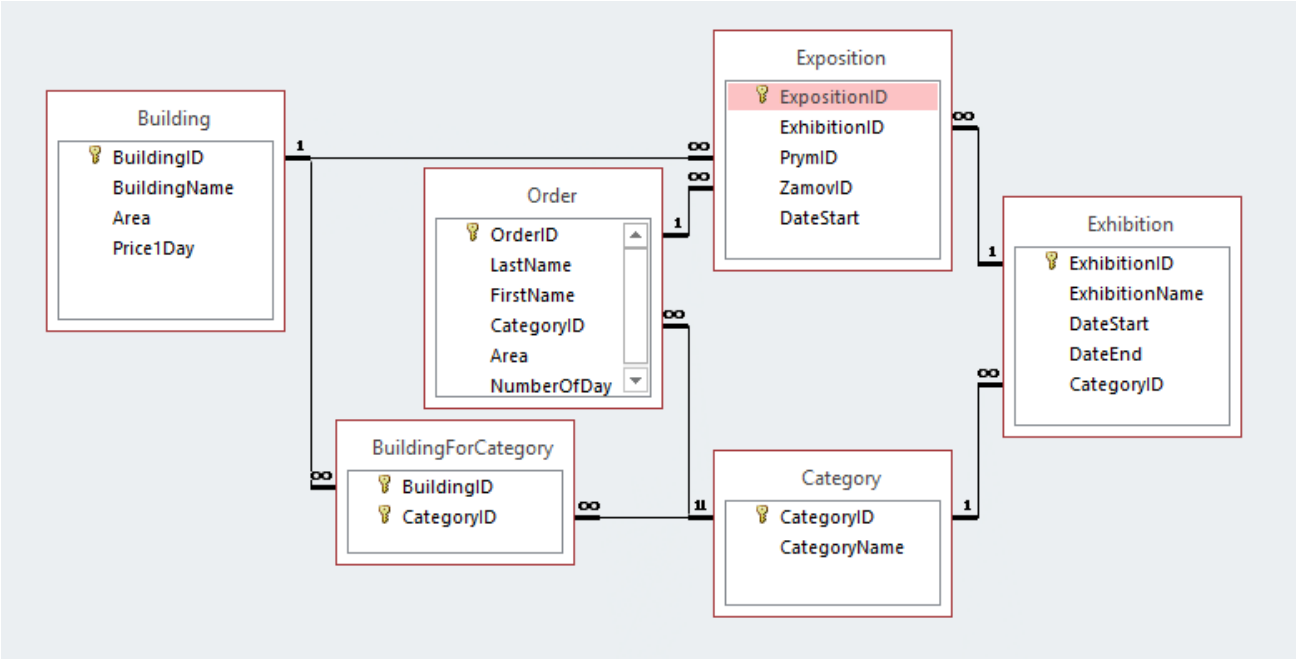

Рисунок 3.1 — База даних в Microsoft Access

Основна сутність це Експозиція (Exposition) вона має 3 зовнішніх ключа які служать не інакше як зв’язками між таблицею Експозиція (Exposition) та таблицями Приміщення (Building), Замовлення (Order) та Виставка (Exhibition). В основній таблиці Атрибути мали назви відповідо: BuildingId, OrderId та ExhibitionId. Тип цих трьох зовнішніх ключів — цілочисельний (int). Також в сутності замовлення є такі атрибут як ДатаПочатку (DateStart), з типом Дата (DateTime). І останні атрибути — це атрибути які є у більшості сутностей, вони називаються: Ключ (Id), з типом даних цілочисельний (int), Примітка (Note) та Опис (Descriptions), з типом текст (string).

Сутність Категорія (Category) є дочірньою для сутностей Замовлення (Order) і Виставка (Exhibition). Атрибути цієї сутності: первинний ключ Id типу цілочисельний (int), НазваКатегорії (CategoryName), Примітка (Note) та Опис (Description) типу текст (string). Також має зв’язок багато до багатьох з сутністю Приміщення (Building). Цей зв’язок забезпечує сутність КатегоріяПриміщень (BuildingForCategory), яка не містить свій базовий ключ та має 2 зовнішніх ключі КлючПриміщення (BuildingId), КлючКатегорії (CategoryId).

Сутність Виставка (Exhibition) є батьківською для сутності Категорія (Category) і дочірньою для сутності Експозиція (Exposition). Атрибути цієї сутності: первинний ключ Id, зовнішній ключ КлючКатегорії (CategoryId) з типом цілочисельні (int), ДатаПочатку (DateStart) та ДатаЗакінчення (DateEnd) з типом Дата (DateTime), Примітка (Note) та Опис (Description) типу текст (string).

Сутність Приміщення (Building) є дочірньою для сутності Експозиція (Exposition) та ма зв’язок багато до багатьох з сутностю Категорія (Category) . Атрибути цієї сутності: первинний ключ Id, Площа (Area) та ЦінаЗа1День (Price1Day) типу int, Назва (BuildingName), Примітка (Note) та Опис (Description) типу текст (string).

Сутність Замовлення (Order) є дочірньою для сутності Експозиція (Exposition) та батьківською для сутності Категорія (Category) . Атрибути цієї сутності: первинний ключ Id, зовнішній ключ КлючКатегорії (CategoryId),

Площа (Area) та КількістьДнів (NumberOfDay) типу цілочисельні (int), Назва (BuildingName), Примітка (Note) та Опис (Description) типу текст (string).

3.2 Реалізація сховища даних

Репозиторій — це місце, де зберігаються й підтримуються деякі дані. В даному випадку репозиторій — це місце де зберігається моя XSD-схема та XML-файл (Додаток Д).

Для того, щоб створити проєкт, потрібно після запуску Microsoft Visual Studio 2019 зайти у меню: Файл — Створити — Проєкт. Зліва у меню вибрати мову — C#, платформу — Windows, у основній частині вибирати Бібліотека класів, натиснути кнопку Далі (рис. 3.2). В полі Ім’я проєкту задати ім’я бібліотеки класів, в даному випадку це SalonCL, зазвичай поля ім’я рішення стає таким як і ім’я бібліотеки класів, але його перейменовуємо в Salon. натиснути кнопку Створити (рис. 3.3).

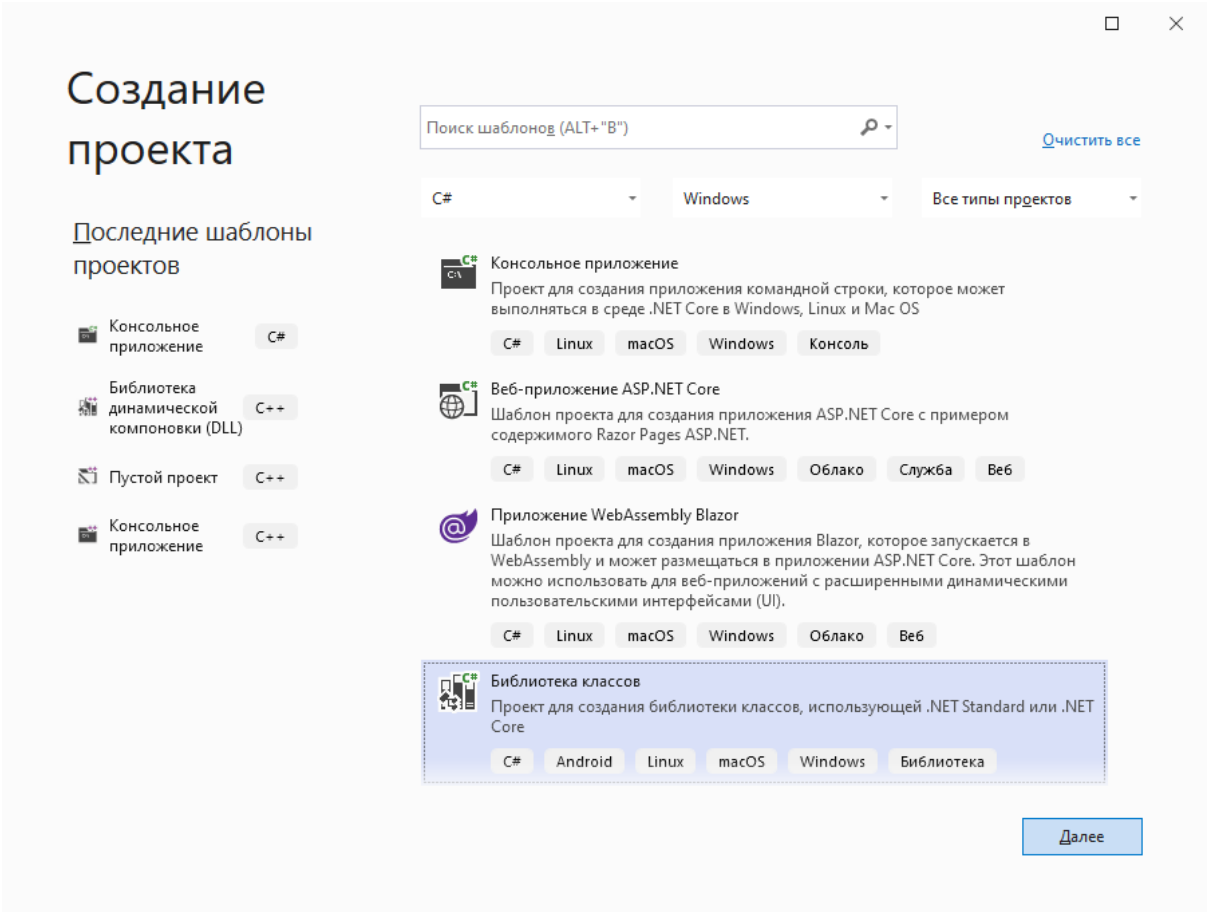

Рисунок 3.2 — Створення бібліотеки класів

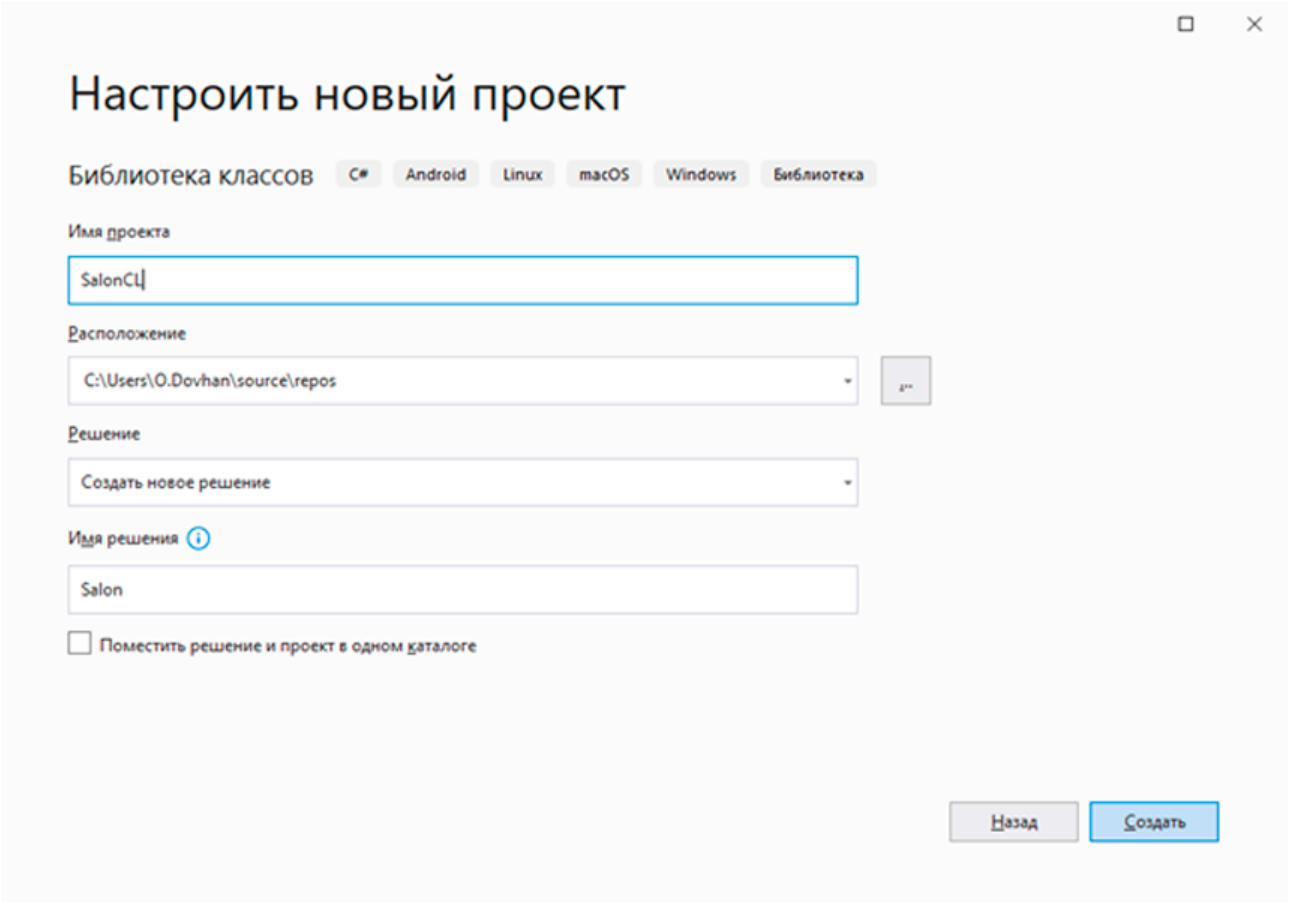

Рисунок 3.3 — Створення бібліотеки класів

Після створення Бібліотеки класів створюємо в ній папку з назвою Repository для цього потрібно клікнути по бібліотеці правою кнопкою миші вибрати пункт Добавити — Створити папку. Далі додано в цю папку клас, для цього треба клікнути по папці Repository правою кнопкою миші, вибрати Добавити—Клас і задати йому ім’я EntityRepository.cs, в цьому класі буде реалізований метод отримання шляху до файлу (лістинг 3.1)

Лістинг 3.1 — Отримання шляху до файлу

```
private string filePath;                    //оголошення змінної типу string
private string directory;                   //оголошення змінної типу string
public string Directory
{
   get { return directory; }                        //повертає шлях до папки
   set {if (String.IsNullOrWhiteSpace (value))
```

```
    {
        //якщо шлях до папки null або пробіли то в exeName
        //записуємо шлях до збірки
        string exeName = Assembly.GetExecutingAssembly ().Location;
        //записуємо у directory шлях до папки Repository що знаходиться в
        //збірці
        directory =
            Path.GetDirectoryName (exeName) + "\\Repository";
            // видаляємо пробіли спочатку і в кінці
        }else directory = value.Trim ();
        //перевірка чи існує така папка якщо ні то створюємо її
        if (!File.Exists (directory))
            System.IO.Directory.CreateDirectory (directory);
        filePath = directory + "\\";              //шлях до файлу
    }
}
```

Наступний крок — створити схему XSD. Для цього нам треба створити DataSet, заповнити його об’єктами DataTable (лістинг 3.2), які в свою чергу заповнені атрибутами і створити зв’язки між цими таблицями (лістинг 3.3).

Лістинг 3.2 — Отримання шляху до файлу

```
public override DataSet CreateDataSet ()          //оголошення методу
    {
     //створення екземпляра об’єкта DataSet та задання імені
     DataSet ds = new DataSet("ExpositionRepository");
     //створення сутностей
     ds.Tables.Add(CreateExpositionTable());
     ds.Tables.Add(CreateExhibitionTable());
     ds.Tables.Add(CreateBuildingTable());
     ds.Tables.Add(CreateCategoryTable());
```

```
 ds.Tables.Add(CreateOrderTable());
 ds.Tables.Add(CreateBuildingForCategoryTable());
 CreateRelations (ds);       //створення зв’язків між сутностями
 return ds;         // повертаємо ds
}
```

Лістинг 3.3 — Створення сутностей

```
public DataTable CreateCategoryTable ()    //оголошення методу
      {
 //створення екземпляру об’єкта DataTable та задання імені
 DataTable dt = new DataTable("Category");
 AddIdColumn(dt);                //додаємо ключове поле Id
 DataColumn dc;                  //оголошуємо змінну типу DataColumn
 //створюємо новий об’єкт задаючи ім’я та тип данних
 dc = new DataColumn("CategoryName", typeof(string));
 dt.Columns.Add(dc);             //додаємо цей об’єкт до DataTable
AddNoteColumn(dt);                      //додаємо атрибут Note
AddDescribeColumn(dt);                        //додаємо атрибут Description
 return dt; //повертаєм значення
 }
```

Створивши таблиці та заповнивши їх атрибутами можна перейти до створення зв’язків між таблицями. Для того, щоб з’єднати дві таблиці у них повинне бути спільне поле. Причому, одна таблиця буде являтись батьківською, а друга дочірньою (лістинг 3.4)

Лістинг 3.4 — Створення зв’язків

```
//оголошуємо зв’язок
DataRelation drl;
//створюємо об’єкт
drl = new DataRelation("Category_Exhibition",
//вказуємо таблицю та атрибут в ній
```

```
ds.Tables["Category"].Columns["Id"],
//вказуємо таблицю та атрибут в ній
 ds.Tables["Exhibition"].Columns["CategoryId"]);
 ds.Relations.Add(drl);    //додаємо даний об’єкт до DataSet
```

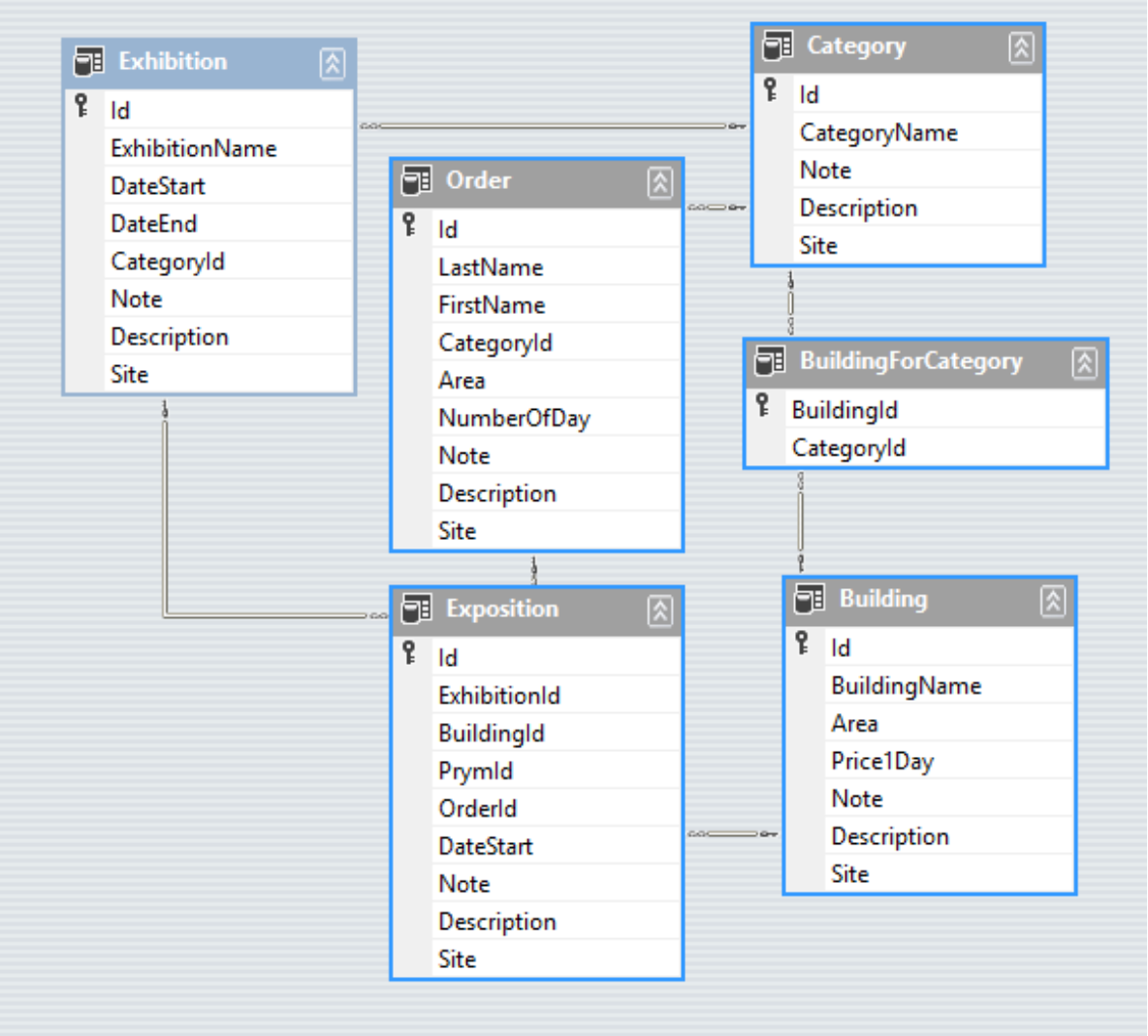

Рисунок 3.4 — Схема даних

Створивши всі таблиці в DataSet, отримуємо схему даних (рис. 3.4) та схему класів (Додаток Б).

3.3 Реалізація інтерфейсу введення даних

Користувацький графічний інтерфейс є важливою складовою частиною більшості програмних додатків. Від нього залежать зручність та швидкість користування програмою.

Однією з переваг використання Windows Forms для розробки інтерфейсу є зручне моделювання. Для цього потрібно лише обрати потрібний елемент керування із панелі інструментів та розмістити його у потрібному місці форми.

Для створення нової форми потрібно зайти у Micrisoft Visual Studio, обрати пункт меню Проєкт — Додати форму (Windows Forms), після чого відкриється вікно (рис. 3.5) де вже буде обраний пункт Windows Form і залишиться тільки вказати потрібне ім’я форми.

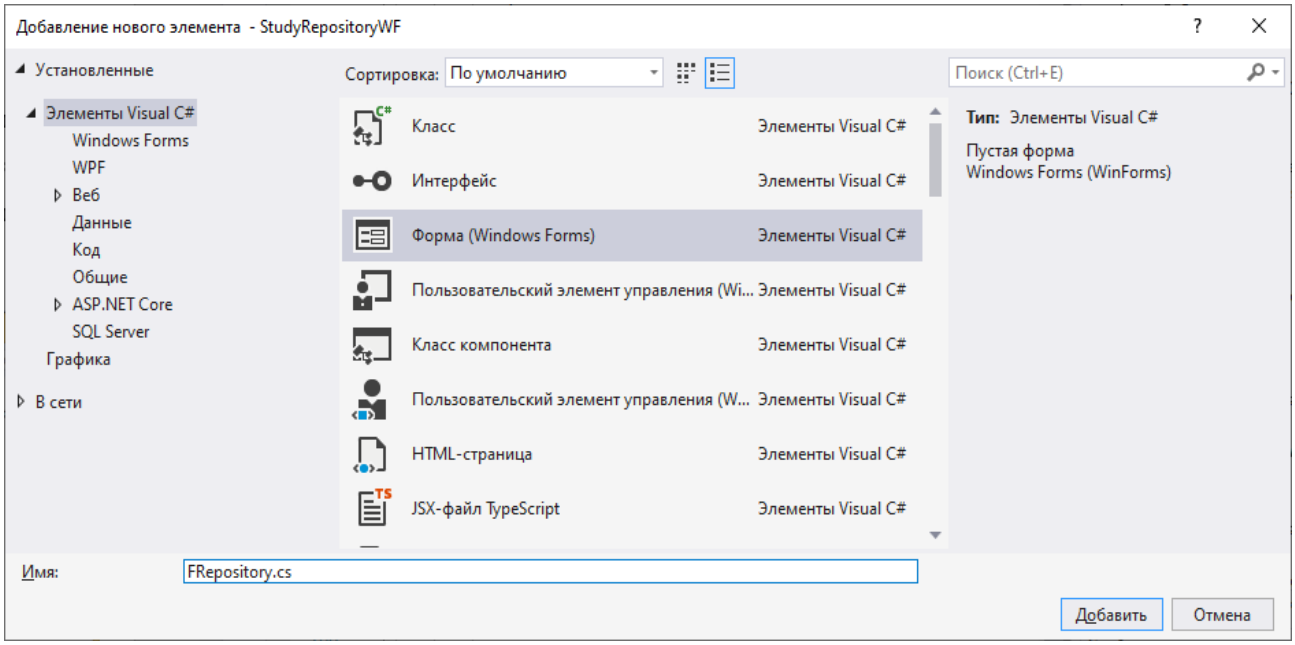

Рисунок 3.5 — Створення форми Windows Forms

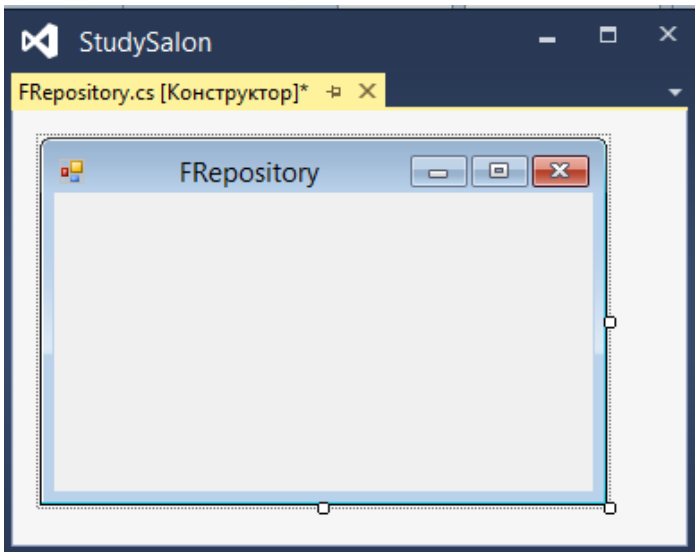

Рисунок 3.6 — Пуста форма FRepositary

Створено основне вікно, назвавши його FRepositary. Після створення, отримано пусту форму (рис. 3.6), на якій надалі розміщено елементи керування для надання користувачеві інтерфейсу та реалізації роботи функцій.

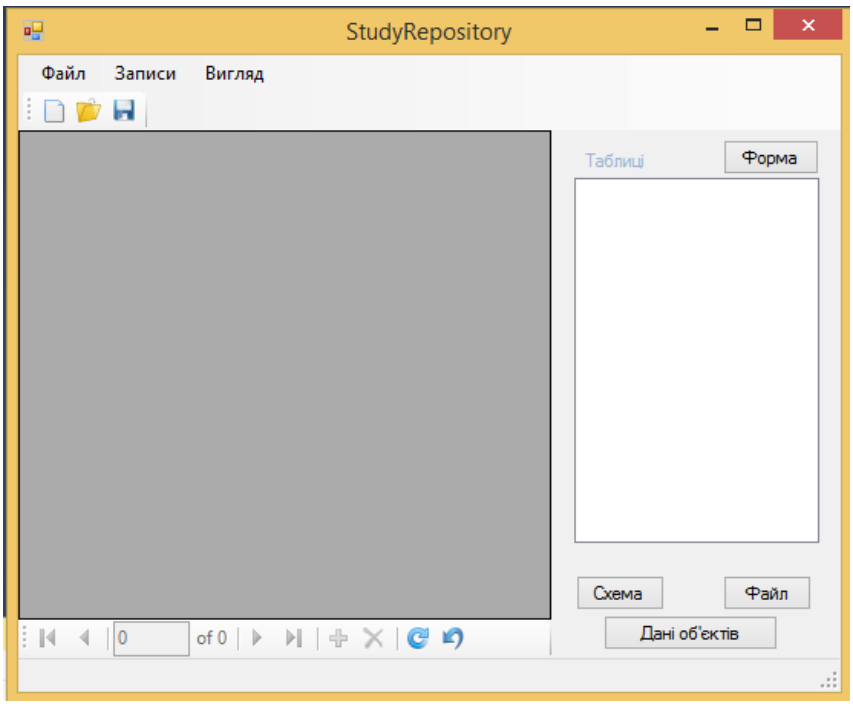

Рисунок 3.7 — Готова FRepositary форма

Зробивши форму відповідних розмірів додано такий набір елементів:

— MenuStrip — система меню для форми;

— ToolStrip — контейнер для об’єктів панелі інструментів;

— TableLayoutPanel — область, в якій вміст динамічно відображається в таблиці, що складається із стрічок та стовпців;

— FlowLayoutPanel — область, в якій вміст динамічно відображається вертикально або горизонтально;

— Lable — надпис;

— TextBox — текстове поле;

— Button — кнопка;

— RichTextBox — елемент керування з розширеним редагування для роботи з об’єктами FlowDocument;

— ToolStripButton — кнопка контейнеру ToolStrip;

— Combobox — елемент керування, призначений для вибору значень із списоку, що розкривається;

— Panel — елемент керування, призначений для розміщення в ньому інших елементів керування;

— StatusStrip — «стрічка стану».

Після чого отримаємо таку форму (рис. 3.7)

Далі треба було створити форму, яка буде допомагати користувачу більш просто й детально заповнювати таблиці необхідною інформацією. Для зручності, спочатку створюємо BaseTableForm (базову форму) та EntityTableForm (успадковується від базової) та прописуємо усі необхідні методи над об’єктами форм, щоб вони містились у всіх подальших формах. Щоб додати форму Windows Forms, необхідно: Додати — Клас — Форма Windows Forms — Назва форми «BaseTableForm» — Додати. На робочому полі з’явиться відповідна форма. Так само додається і форма «EntityTableForm».

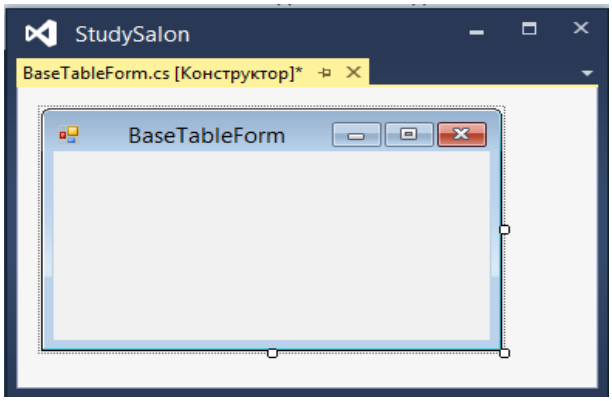

Рисунок 3.8 — Базова Форма

Після створення BaseTableForm (рис. 3.8), додаємо на неї bindingNavigator та додаткові кнопки для керування записами. Задаємо властивості Anchor значення Bottom. Отримуємо таку форму, яку будем використовувати як базову.

Кожна наступна форма буде наслідувати дану базову форму. Так як на даній формі майже немає ніяких елементів, всі вони будуть знаходитись на дочірніх формах. Щоб створити наслідовану форму необхідно натиснути Проєкт — Додати форму Windows та вибрати похідна форма, назвати

новостворену форму, та унаслідувати нашу базову форму BaseTableForm (рис. 3.9).

Отримуємо нову форму, яка початково аналогічна нашій базовій формі. Додаємо на неї необхідні нам елементи:

— Panel — елемент керування, призначений для розміщення в ньому інших елементів керування;

— Lable — надпис;

— TextBox — текстове поле;

— Combobox — елемент керування, призначений для вибору значень із список, розкривається.

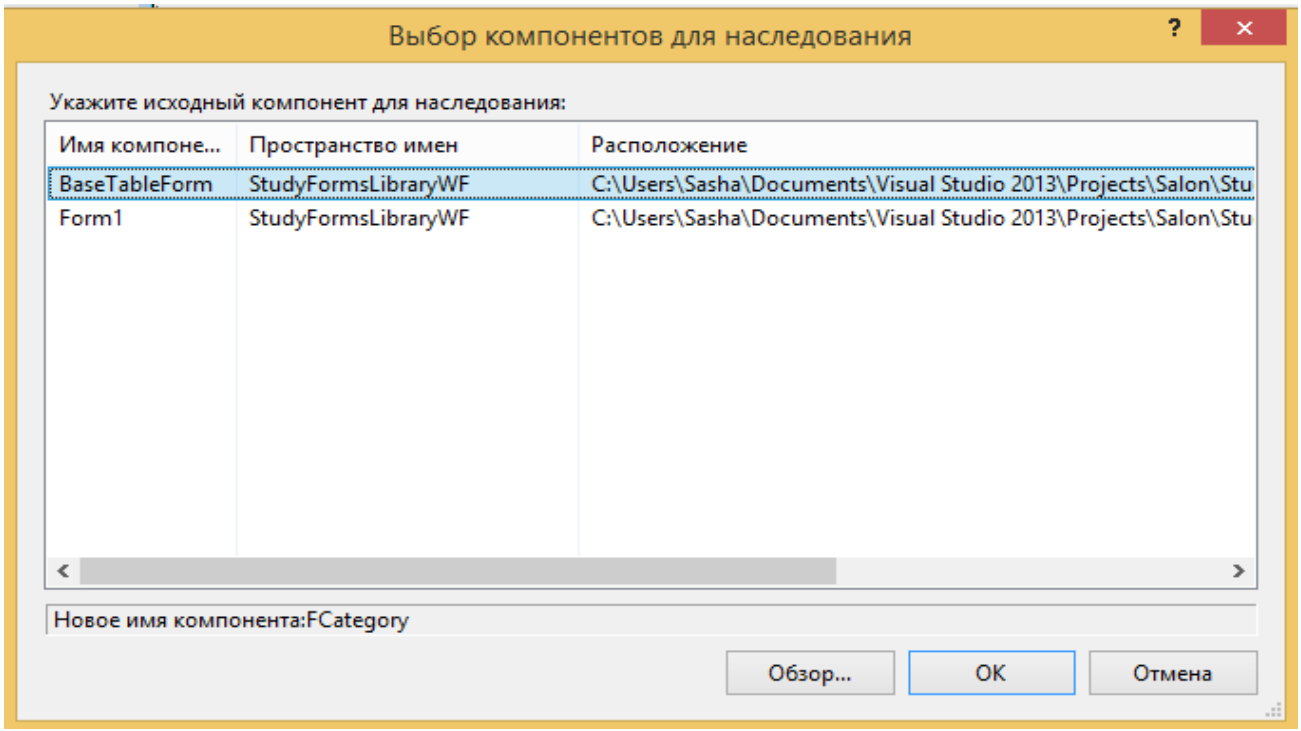

Рисунок 3.9 — Вибір базової форми

Отриманий результат зображений на (рис 2.10)

Збереження інформації до DataSet показана в Лістингу 3.5

Лістинг 3.5 Збереження інформації

```
protected override void SetDataBinding()
{ //перевизначаємо метод SetDataBinding
```

```
base.SetDataBinding();//викликаємо базовий метод SetDataBinding
InitializeComponent();//виконуємо ініціалізацію компонентів
//очищаємо елемент txtName
txtName.Text = "";
//робим прив’язку тексту з елемента txtName до атрибуту CategoryName
txtName.DataBindings.Add(new Binding("Text",
        bindingSource, "CategoryName", true));
}
```

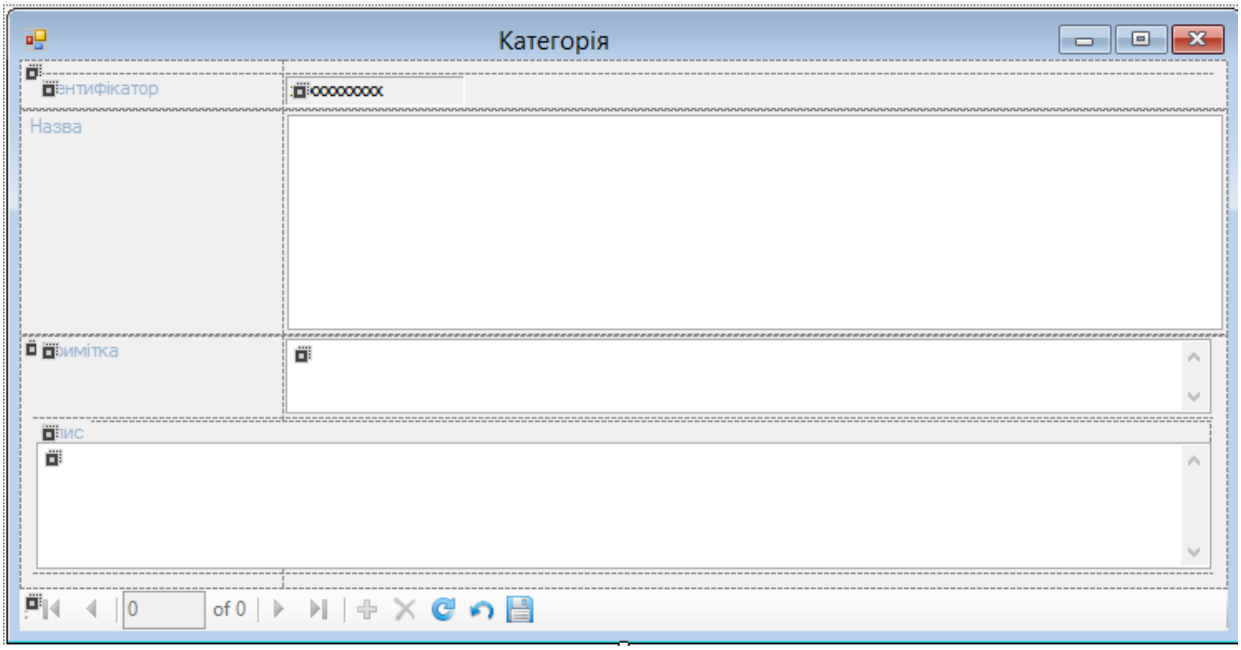

Рисунок 3.10 — Інтерфейс отримання інформації

Схему ієрархії класів форм можна переглянути в додатку В.

3.4 Реалізація класів предметної області

Кожна таблиця містить поле ідентифікації (Id). Тому виносимо його в базовий клас Entity, який потім наслідують похідні класи. Похідні класи містять свої поля, реалізовані як автоматичні властивості, які будуть залежати від атрибутів сутностей. Наприклад клас Категорія буде мати такий вигляд

Лістинг 3.6 — Клас Категорія

```
public class Category : DescribedIdentiedNamedEntity
```

```
{
    public string CategoryName { get; set; } // назва категорії
    //оголошення списку посилань на Exhibition
    public List<Exhibition> Exhibitions { get; set; }
    //оголошення списку посилань на Order
    public List<Order> Orders { get; set; }
    //оголошення списку посилань на Building
    public List<Building> Buildings { get; set; }
    public override string ToString() //перевизначення ToString
     {
    // строка що виводиться для кожного елемента з списку
    return String.Format(" \n \t  \"{0}\"  ", CategoryName);
    }
 }
```

Метод ToString() ми перевизначаємо для виведення інформації в графі об’єктів. Даний метод потрібен для виведення на екран всієї інформації про клас.

Схему класів предметної області можна переглянути в додатку Г.

Клас Entity являється абстрактним, тому його екземпляр створювати не можна. Всі його поля та методи, якщо такі є доступні в дочірніх класах.

3.5 Реалізація коду для створення і відображення графа об’єктів

Далі в класі ExpositionRepository створено колекції, в яких надалі буде зберігатись інформація з DataSet, так як це необхідно для відображення графа об’єктів. Приклад створення колекцій можна побачити в лістингу 3.7.

Лістинг 3.7 Створення колекцій

```
 Dictionary<int, Exposition> expositions
     = new Dictionary<int, Exposition>();
 Dictionary<int, Exhibition> exhibitions
```

```
    = new Dictionary<int, Exhibition>();
List <BuildingForCategory>buildingForCategorys
    = new List <BuildingForCategory>();
```

Як видно з лістингу 3.7 остання колекція buildingForCategorys це не словник а список. Це пояснюється тим що даний клас BuildingForCategory не має власного Id.

Далі необхідно створити метод для запису інформації в колекції. Для кожної таблиці має бути окремий метод. Розглянемо збереження на одному з випадків — таблиці Замовлення.

Лістинг 3.8 — Запис інформації в словники

```
private bool FillOrders(DataSet ds)
{
    if (!ds.Tables.Contains("Order"))   //якщо не існує таблиця Order
       return false;   // повернути значення false
    if (this.orders == null)          //якщо колекція orders відсутня
        {
        orders =  new Dictionary<int, Order>();   // створити нову колекцію
        }
  //цикл для кожного запису в таблиці Order
  foreach (DataRow dr in ds.Tables["Order"].Rows)
        {
              Order obj =  new Order();();     //створюєм об’єкт Персони
         //викликаємо метод який заповнює поля об’єкта що знаходяться в
         //базовому класі
         FillEntityMembers(dr, obj);
         //записуємо в поле класа вміст стрічки LastName
         obj.LastName = dr["LastName"].ToString();
         //записуємо в поле класа вміст стрічки FirstName
         obj.FirstName = dr["FirstName"].ToString();
```

```
            //записуємо в поле класа вміст стрічки Area
            obj.Area = Double.Parse(
            dr["Area"].ToString().Replace('.', ','));
            //додаємо до словника поле Id
            this.orders[obj.Id] = obj;
            //записуємо в поле класа вміст стрічки NumberOfDay
            obj.NumberOfDay = int.Parse(
            dr["NumberOfDay"].ToString().Replace('.', ','));
            //записуємо в поле класа вміст стрічки CategoryId
            obj.CategoryId = Int32.Parse(
            dr["CategoryId"].ToString());
        }
      return true; //повернення значення true
    }
```

Оскільки вже маємо колекцію об’єктів, то тепер можна приступити до виведення її на екран. Інформація буде виводитись на спеціальну форму. Єдиний елемент цієї форми — RichTextBox.

Наступний наш крок це вказати елементу, як працювати з текстом.

Лістинг 3.9 — Налаштування RichTextBox

```
//якщо об'єкт пустий чи не створений,
if (String.IsNullOrWhiteSpace(value))
  {
        // в заголовку форми пишемо "Файл: (не задано)"
    this.Text = "Файл: (не задано)";
    //в самій формі — нічого не виводимо
    richTextBox1.Text = "";
    }
else            //інакше
    {
```

```
//в заголовку пишемо шлях до файлу і завантажуємо в елемент
//керування  відповідний тип файлу, який вказує типи потоків даних
this.Text = "Файл: " + value;
richTextBox1.LoadFile(value, RichTextBoxStreamType.PlainText);
}
```

Далі привязуємо виклик форми до кнопки «Схема» на головній формі. (лістинг 3.10)

Лістинг 3.10. — Прив’язка форми до кнопки.

```
private void btnShcema_Click(object sender, EventArgs e)
{
    FTextInfo frm = new FTextInfo ();     //створюємо новий екземпляр
    frm.Title = "Граф об'єктів сховища даних";         //задаємо назву
    //задаємо вміст
    frm.DataString = controller.RepositoryObjectsAsString;
    frm.ShowDialog ();         //викликаєм форму
}
```

Запустивши програму і натиснувши на кнопку з’являється вікно зображене на рисунку 3.11

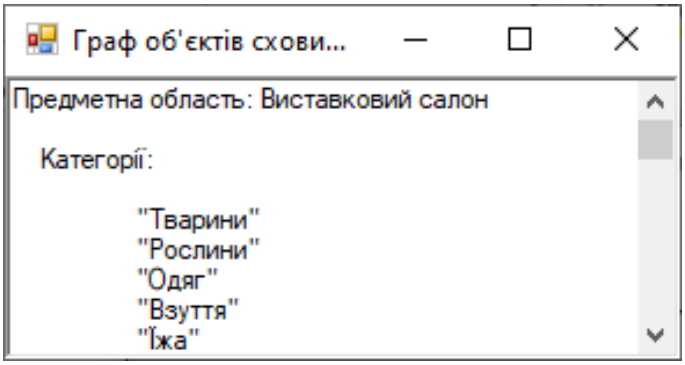

Рисунок 3.11 — Граф об'єктів сховища даних

Тепер треба перевизначити метод ToString в ExpositionRepository.

Лістинг 3.11 Перевизначення методу ToString в класі ExpositionRepository

```
  public override string ToString()
{
   // сворення нової змінної  —  строки символів
   StringBuilder sb = new StringBuilder();
   sb.Append("Предметна область: Виставковий салон"); //перший рядок
   sb.Append(Categorys.ToStringExt("\n    Категорії"));    //другий рядок
   sb.Append(Buildings.ToStringExt("\n    Приміщення"));
   sb.Append(Exhibitions.ToStringExt("\n    Виставки"));
   sb.Append(Expositions.ToStringExt("\n    Експозиції"));
   sb.Append(Orders.ToStringExt("\n    Замовлення"));

   return sb.ToString();     //повертає рядок
}
```

У лістингу 3.11 масиви створені на основі відповідних колекцій.

Лістинг 3.12 Створення масиву на основі колекції Dictionary

```
  public Order[] Orders
{
   get
   {
      return orders
            .Select(e => e.Value)
            .ToArray();
   }
}
```

Виконуючу над даним масивом метод розширення ToStringExt , який в свою чергу перебирає всі елементи масиву і вже над ними виконує метод ToString. Щоб отримати коректний результат потрібно перевизначити метод ToString в даному класі предметної області.

Лістинг 3.13 — Перевизначення методу ToString в класах сутностей

```
public override string ToString()
        {
            return String.Format
            (" \n \t  {0} {1} \n \t  Площа:  {2}\n \t Кількість днів: {3} \n\t
                                                        Категорія: {4} \n ",
             LastName, FirstName, Area, NumberOfDay,
             Category.CategoryName );
         }
```

Клас предметної області містить прямі посилання на екземпляри класу Категорія. Ці зв’язки створюємо в окремих класах. (лістинг 3.14)

Лістинг 3.14 — Створення зв’язків з категорією

```
private void CreatOrderLinks()
          {
            if (orders == null) // якщо колекція orders рівна 0
              return; // повторюємо
                  //переносимо дані з колекції у масив
                  Order[] orderArray =
                     orders.Select(e => e.Value)
                     .ToArray();
             for (int i = 0; i < orderArray.Length; i++)        //цикл
             {
                //якщо масив не рівний 0
                 if (orderArray[i].CategoryId != 0)
                 {
                          orderArray[i].Category = categorys[
                             orderArray[i].CategoryId];
                          orderArray[i].Category.Orders.Add(orderArray[i]);
                 }
```

```
        }
    }
```

Тепер ми можемо користуватися в методі ToString класу Order прямими посиланнями на об’єкти певних класів. Як показано в лістингу 3.13 ми використовуємо посилання на Категорію для отримання доступу до їх полів.

Результат зображений на рисунку 3.12

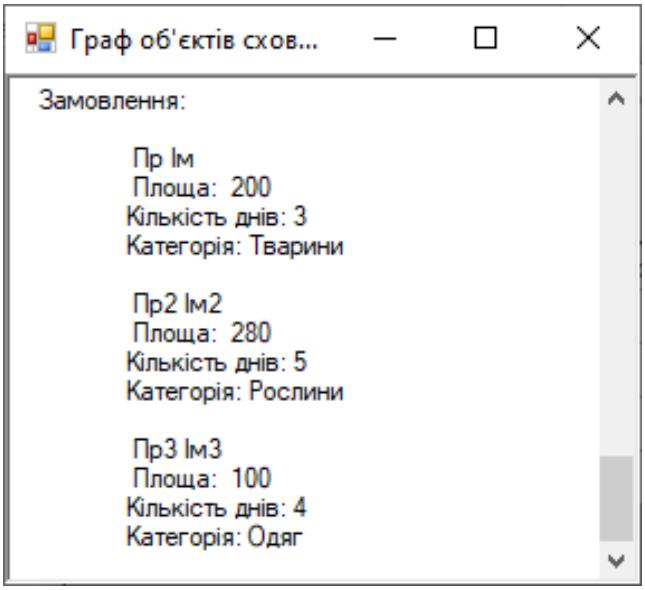

Рисунок 3.12 — Граф об’єкта Order

3.6 Реалізація вибіркового відновлення

Так як колекції, в яких надалі буде зберігатись інформація з DataSet, вже створені, далі необхідно створити метод для запису вибіркової інформації в колекції. Для кожної таблиці має бути окремий метод. Розглянемо збереження вибіркової інформації з таблиці Експозиція (лістинг 2.15).

Лістинг 3.15 — Запис вибіркової інформації в словники

```
private bool FillSelectExpos(DataSet ds)
{
  if (!ds.Tables.Contains("Exposition")) //якщо не існує таблиця Exposition
     return false; // повернути значення false
  if (this.expositions == null) //якщо колекція expositions відсутня
```

```
{
  // створити нову колекцію
   expositions = new Dictionary<int, Exposition>();
}   //цикл для кожного запису в таблиці Exposition
foreach (DataRow dr
   in ds.Tables["Exposition"].Rows)
{
   Exposition obj =
      new Exposition(); //створюєм об'єкт Експозиції
   //викликаємо метод який заповнює поля об'єкта що знаходяться в
   //базовому класі
   FillEntityMembers(dr, obj);
   //записуємо в поле класа вміст стрічки ExhibitionId
   obj.ExhibitionId = Int32.Parse(
      dr["ExhibitionId"].ToString());
   //записуємо в поле класа вміст стрічки ExpositionName
   obj.ExpositionName =
     dr["ExpositionName"].ToString();
   //записуємо в поле класа вміст стрічки BuildingId
   obj.BuildingId = Int32.Parse(dr["BuildingId"].ToString());
   //записуємо в поле класа вміст стрічки OrderId
   obj.OrderId = Int32.Parse(dr["OrderId"].ToString());
   //додаємо до словника поле Id
   this.expositions[obj.Id] = obj;
   //записуємо в поле класа вміст стрічки DateStart
   obj.DateStart = DateTime.Parse(dr["DateStart"].ToString());
   if (obj.DateStart > DateTime.Today) //якщо не існує запис
                                       //в таблиці Exposition
                                       //з полем DateStart сьогодні
      return false;// повернути значення false
```

```
        }
        return true; //повернення значення true
    }
```

Оскільки вже маємо колекцію об’єктів, то тепер можна приступити до виведення її на екран. Інформація буде виводитись на спеціальну форму. Єдиний елемент цієї форми — RichTextBox (лістинг 3.16).

Наступний наш крок це вказати елементу, як працювати з текстом.

Лістинг 3.16 — Налаштування RichTextBox

```
//якщо об'єкт пустий чи не створений,
if (String.IsNullOrWhiteSpace(value))
  {
        // в заголовку форми пишемо "Файл: (не задано)"
    this.Text = "Файл: (не задано)";
    //в самій формі — нічого не виводимо
    richTextBox1.Text = "";
    }
else            //інакше
    {
    //в заголовку пишемо шлях до файлу і завантажуємо в елемент
    //керування  відповідний тип файлу, який вказує типи потоків даних
    this.Text = "Файл: " + value;
    richTextBox1.LoadFile(value, RichTextBoxStreamType.PlainText);
    }
```

Далі привязуємо виклик форми до кнопки «Заплановані виставки» на головній формі. (лістинг 3.17)

Лістинг 3.17 — Прив’язка форми до кнопки.

```
private void btnExhib_Click (object sender, EventArgs e)
{
```

```
            FTextInfo frm = new FTextInfo ();     //створюємо новий екземпляр
            frm.Title = "Заплановані виставки";  //задаємо назву
            //задаємо вміст
            frm.DataString = controller.RepositoryObjectsAsString;
            frm.ShowDialog ();          //викликаєм форму
        }
```

Запустивши програму і натиснувши на кнопку з’являється вікно зображене на рисунку 3.13

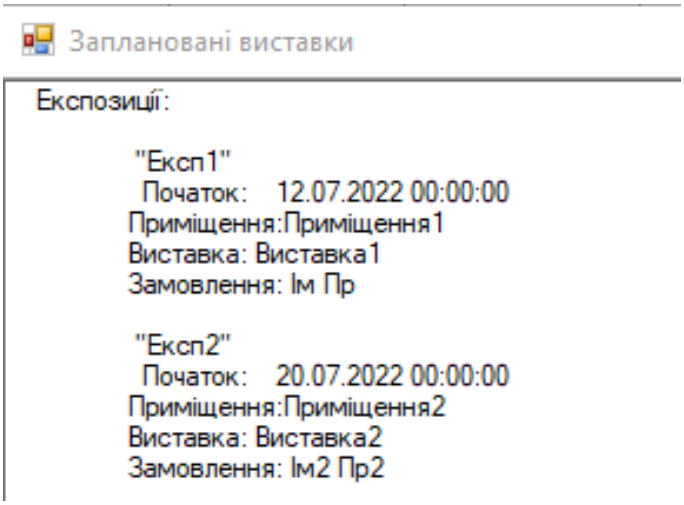

Рисунок 3.13 — Заплановані виставки

ВИСНОВКИ

У ході виконання бакалаврської дипломної роботи:

— проаналізовано предметну область;

— реалізовано сховище даних;

— реалізовано інтерфейс введення даних;

— реалізовано класи предметної області;

— реалізовано код для створення і відображення графа об’єктів;

— реалізовано вибіркове відновлення.

Розроблена програма, яку може використовувати будь-який користувач, якому потрібно робити облік для виставкового салону, з використанням зручного, простого та швидкого інтерфейсу з надійним сховищем даних.

Сховище даних взаємодіє словниками даних та з об’єктом DataSet. Якщо його заповнити даними, то з’явиться можливість отримати граф об’єктів, завдяки якому можна визначити правильність роботи програми.

В ролі інтерфейсу між користувачем та програмою був створений віконний додаток.

Ще була створена перевірка, яка враховує всі нюанси, помилки та недоліки які може допустити користувач програмного продукту під час роботи із ним.

Робота реалізована мовою C# в середовищі програмування Microsoft Visual Studio.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. О. І. Черняк О. В. Довгань / підсистема збереження даних обліку виставкового салону з вибірковим відновленням //Тези доповіді. Всеукраїнська науково-практична інтернет-конференція Молодь в науці: дослідження, проблеми, перспективи. Вінниця 2022 р. Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2022/paper/view/14566

2. Руководство по программированию C# — [Електронний ресурс] — Режим доступу: http://msdn.microsoft.com/ru-ru/library/67ef8sbd.aspx

3. C Sharp — [Електронний ресурс] — Режим доступу: https://uk.wikipedia.org/wiki/C_Sharp/ — C Sharp — Вікіпедія

4. Visual Studio — [Електронний ресурс] — Режим доступу: http://msdn.microsoft.com/ua-ua/hh916378.aspx — Microsoft Visual Studio.

5. Семеренко В.П., Програмування мовами С та С++ в середовищі Windows.– Вінниця: ВДТУ, 2002 р. –128с.

6. Герберт Шилдт. C# учебный курс = C#. — М.: Питер, 2002. — С. 512. — 5-94723-167-0.

7. Windows Forms — [Електронний ресурс] — Режим доступу: http://msdn.microsoft.com/ua-ua/jj244581 — Windows Forms

8. Завадський І.О. Основи баз даних. — Київ: Видавництво ПП І.О. Завадський, 2010. — 192 с.

9. Бондарев В.М. Програмирование. — 2005. — 194 с.

10. Фрэд Лонг та ін. (2014). Руководство для программиста на Java: 75 рекомендаций по написанию надежных и защищенных программ. «Вільямс». ISBN 978-5-8459-1897-0.

11. Брайан Холл. Beej's Guide to C Programming: Пер. с англ / Брайан Холл. — М. : ООО "І.Д. Вильямс", 2011. – 142 с.

12. Lars Powers, Mike Snell, Microsoft Visual Studio 2008 Unleashed, USA, 2008 – 1536с.

13. Герберт Шилдт. C# учебный курс = C#. A Beginner's Guide. — М.: Питер, 2003. — С. 20. — ISBN 966-552-121-7.

14. С. Байдачний, М. Остапчук. Windows 10 для C# розробників — Київ: IT-книга, 2016. – 312 с. – ISBN 978-966-97182-3-5.

15. Троелсен Е. Язык программирования C# 2005 и платформа .NET 2.0 = PRO C# 2005 and the .NET 2.0 Platform : Пер. с англ / Е. Троелсен. — 3-е изд. – М. : ООО "И.Д. Вильямс", 2007. – 1168 с. – ISBN 5-8459-1124-9.

16. Брайан У. Керниган, Деннис М. Ритчи. Beej's Guide to C Programming: Пер. с англ / Е. Троелсен. — 3-е изд. – М. : ООО "И.Д. Вильямс", 2007. – 1168 с. – ISBN: 978-617-7812-80-6.

ДОДАТОК А

Міністерство освіти та науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

Зав. кафедри ОТ, проф., д.т.н.

_____О.Д. Азаров

"____"_____ 2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання бакалаврської дипломної роботи

«Підсистема збереження даних обліку виставкового салону з вибірковим відновленням»

08-23.БДР.005.00.000 ПЗ

Науковий керівник к.т.н. доц. каф. ОТ

_____ Черняк О.І.

Студентка групи КІ-20мсз

_____ Довгань О.В.

Вінниця 2022

1 Підстави для виконання бакалаврської дипломної роботи (БДР)наступні:

— актуальність розробки, яка полягає у можливості оптимізації робочого часу за рахунок підсистеми обліку даних;

— наказ про затвердження теми бакалаврської дипломної роботи.

2 Мета і призначення БДР наступні:

— метою БДР є розробка підсистеми збереження даних обліку виставкового салону з вибірковим відновленням в середовищі Visual Studio 2019, що дозволить структурувати дані та оптимізувати робочий час.

— призначення розробки полягає у виконанні бакалаврської дипломної роботи.

3 Вихідні дані для виконання БДР наступні:

— технічний опис програмного застосунку;

— мова програмування C#;

— віконний додаток;

— середовище розробки Microsoft Visual Studio.

4 Вимога до виконання БДР наступна:

— створення віконного додатку;

— можливість створювати нові записи;

— можливість відновлювати дані.

5 Етапи БДР та очікувані результати: робота виконується за вісім етапів, таблиця А.1.

6 Матеріали, що подаються до захисту БДР:

— пояснювальна записка БДР;

— графічні і ілюстративні матеріали;

— протокол попереднього захисту БДР на кафедрі;

— відгук наукового керівника;

— відгук рецензента;

— анотації до БДР українською та іноземною мовами;

— нормоконтроль про відповідність оформлення ДР діючим вимогам.

Таблиця А.1 — Етапи виконання роботи

№ етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Постановка задачі роботи	14.02.22	14.02.22	Вступ
2	Аналіз сучасного стану техологій	15.02.22	15.03.22	Розділ 1
3	Аналіз предметної області	16.03.22	28.03.22	Розділ 2
4	Реалізація сховища даних інтерфейсу введення даних	29.03.22	08.04.22	Розділ 2
5	Реалізація інтерфейсу введення даних	09.04.2	29.04.22	Розділ 2
6	Реалізація коду для створення і відображення графа об’єктів	30.04.22	27.05.22	Розділ 2
7	Оформлення пояснювальної записки та ілюстративного матеріалу	28.05.22	06.06.22	Пояснювальна записка

7 Порядок контролю виконання та захисту БДР:

— виконання етапів графічної та розрахункової документації ДР контролюється науковим керівником згідно зі встановленими термінами;

— захист ДР відбувається на засіданні Державної екзаменаційної комісії, затвердженою наказом ректора.

8 Вимоги до оформлення БДР:

— вимоги викладені в МЕТОДИЧНИХ ВКАЗІВКАХ до дипломного проектування, ДСТУ_ 3008–2015, ДСТУ 3974–2000 «Правила виконання дослідно–конструкторських робіт;

— загальні положення» та діючого ГОСТ 2.114–95 ЕСКД.

Технічне завдання до виконання отримала _____ Довгань О.В.

ДОДАТОК Б

Схема класів репозиторія

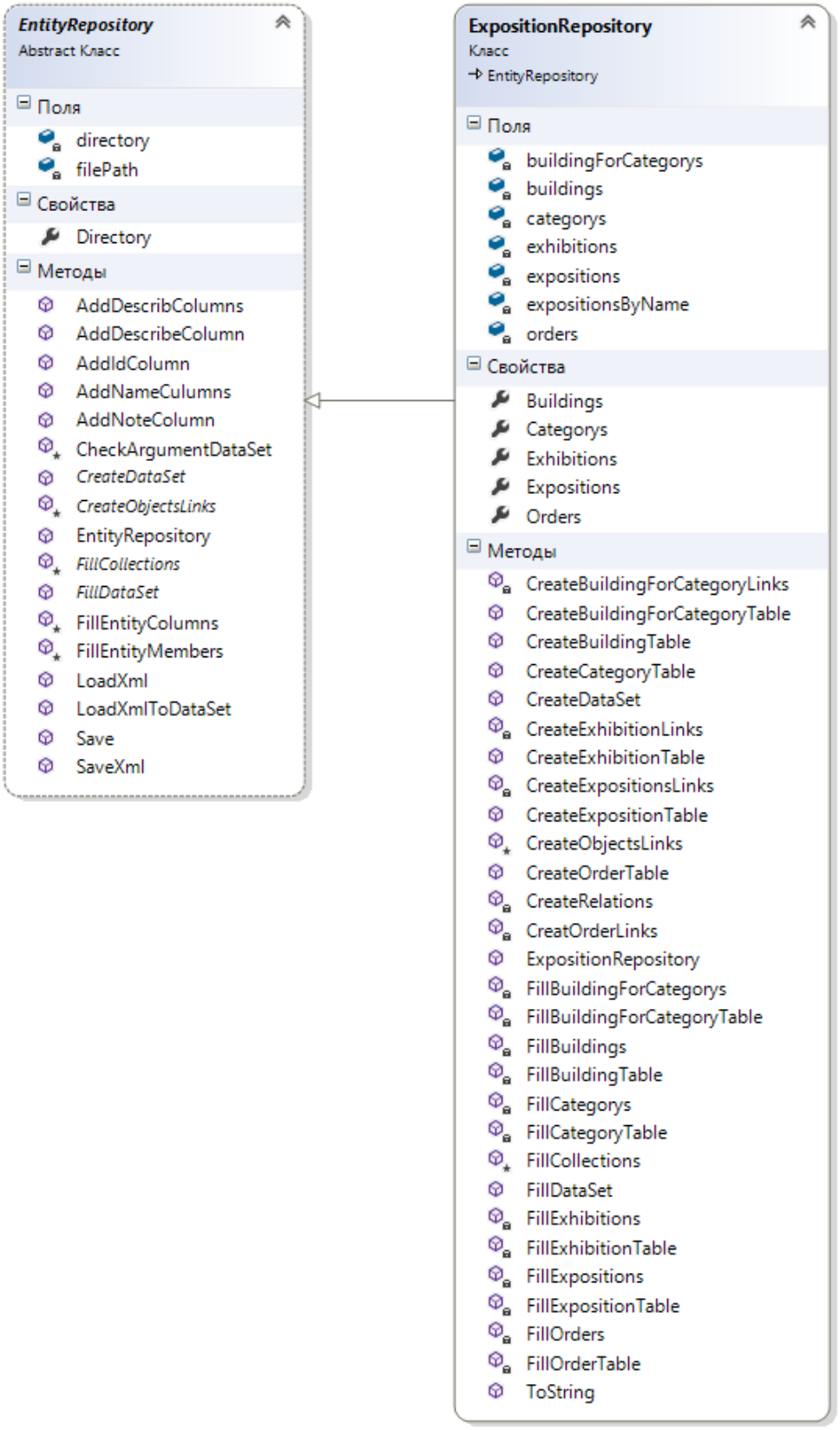

Рисунок Б.1 — Схема класів репозиторія

ДОДАТОК В

Схема ієрархії класів форм

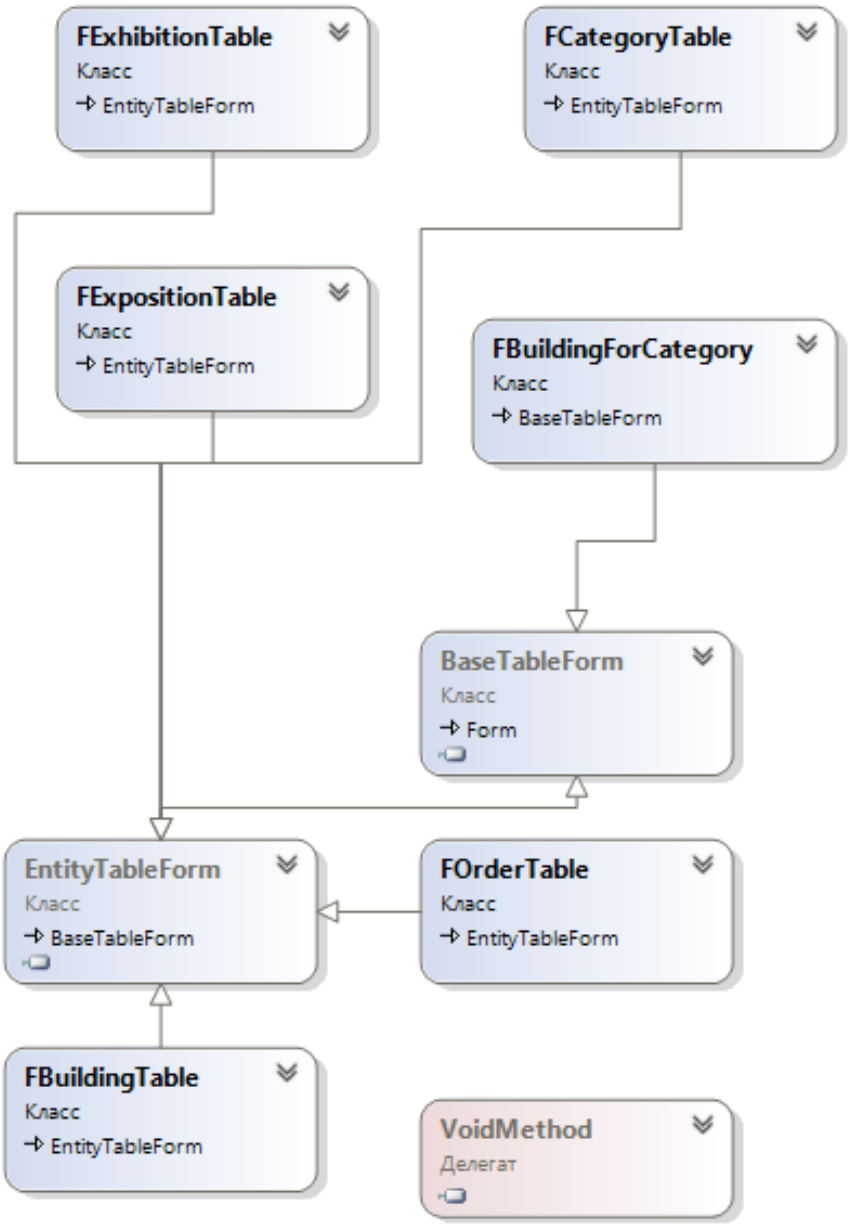

Рисунок В.1 — Схема ієрархії класів форм

ДОДАТОК Г

Схема класів предметної області

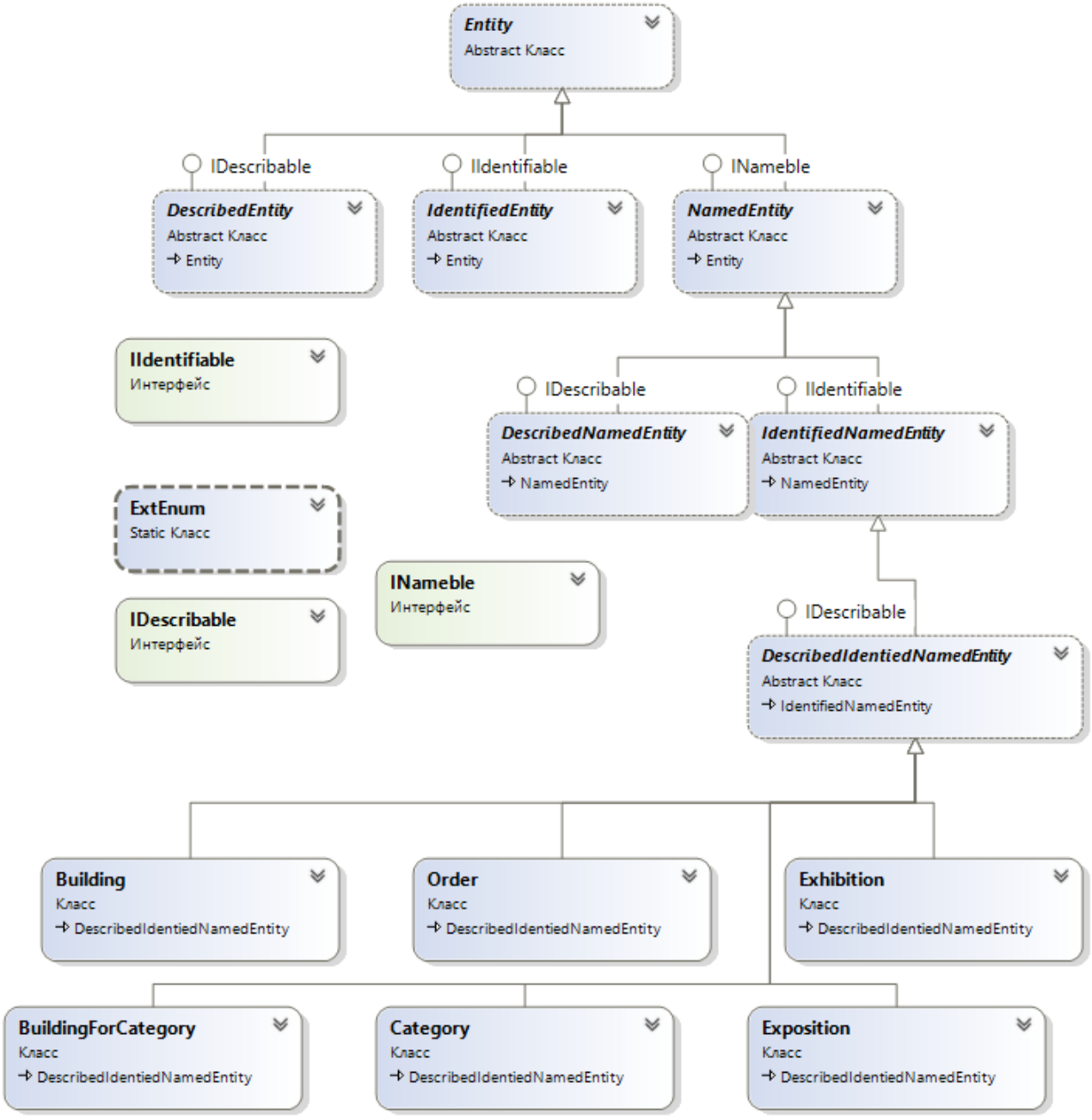

Рисунок Г.1 — Схема класів предметної області

ДОДАТОК Д

XSD-схема

```
<?xml version="1.0" standalone="yes"?>
<xs:schema id="ExpositionRepository" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
  <xs:element name="ExpositionRepository" msdata:IsDataSet="true" msdata:UseCurrentLocale="true">
    <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="Exposition">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Id" msdata:AutoIncrement="true" msdata:AutoIncrementSeed="1" type="xs:int" />
              <xs:element name="ExpositionName" type="xs:string" minOccurs="0" />
              <xs:element name="ExhibitionId" type="xs:int" minOccurs="0" />
              <xs:element name="BuildingId" type="xs:int" minOccurs="0" />
              <xs:element name="OrderId" type="xs:int" minOccurs="0" />
              <xs:element name="DateStart" type="xs:dateTime" minOccurs="0" />
              <xs:element name="Note" type="xs:string" minOccurs="0" />
              <xs:element name="Description" type="xs:string" minOccurs="0" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="Exhibition">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Id" msdata:AutoIncrement="true" msdata:AutoIncrementSeed="1" type="xs:int" />
```

```
            <xs:element name="ExhibitionName" type="xs:string" minOccurs="0" />
            <xs:element name="DateStart" type="xs:dateTime" minOccurs="0" />
            <xs:element name="DateEnd" type="xs:dateTime" minOccurs="0" />
            <xs:element name="CategoryId" type="xs:int" minOccurs="0" />
            <xs:element name="Note" type="xs:string" minOccurs="0" />
            <xs:element name="Description" type="xs:string" minOccurs="0" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Building">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Id" msdata:AutoIncrement="true"
msdata:AutoIncrementSeed="1" type="xs:int" />
            <xs:element name="BuildingName" type="xs:string" minOccurs="0" />
            <xs:element name="Area" type="xs:double" minOccurs="0" />
            <xs:element name="Price1Day" type="xs:decimal" minOccurs="0" />
            <xs:element name="Note" type="xs:string" minOccurs="0" />
            <xs:element name="Description" type="xs:string" minOccurs="0" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Category">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Id" msdata:AutoIncrement="true"
msdata:AutoIncrementSeed="1" type="xs:int" />
            <xs:element name="CategoryName" type="xs:string" minOccurs="0" />
            <xs:element name="Note" type="xs:string" minOccurs="0" />
```

```
            <xs:element name="Description" type="xs:string" minOccurs="0" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Order">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Id" msdata:AutoIncrement="true"
msdata:AutoIncrementSeed="1" type="xs:int" />
            <xs:element name="LastName" type="xs:string" minOccurs="0" />
            <xs:element name="FirstName" type="xs:string" minOccurs="0" />
            <xs:element name="CategoryId" type="xs:int" minOccurs="0" />
            <xs:element name="Area" type="xs:double" minOccurs="0" />
            <xs:element name="NumberOfDay" type="xs:int" minOccurs="0" />
            <xs:element name="Note" type="xs:string" minOccurs="0" />
            <xs:element name="Description" type="xs:string" minOccurs="0" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="BuildingForCategory">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="BuildingId" type="xs:int" minOccurs="0" />
            <xs:element name="CategoryId" type="xs:int" minOccurs="0" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:choice>
  </xs:complexType>
```

```
    <xs:unique name="Constraint1" msdata:PrimaryKey="true">
      <xs:selector xpath=".//Exposition" />
      <xs:field xpath="Id" />
    </xs:unique>
    <xs:unique name="Exhibition_Constraint1"
msdata:ConstraintName="Constraint1" msdata:PrimaryKey="true">
      <xs:selector xpath=".//Exhibition" />
      <xs:field xpath="Id" />
    </xs:unique>
    <xs:unique name="Building_Constraint1" msdata:ConstraintName="Constraint1"
msdata:PrimaryKey="true">
      <xs:selector xpath=".//Building" />
      <xs:field xpath="Id" />
    </xs:unique>
    <xs:unique name="Category_Constraint1" msdata:ConstraintName="Constraint1"
msdata:PrimaryKey="true">
      <xs:selector xpath=".//Category" />
      <xs:field xpath="Id" />
    </xs:unique>
    <xs:unique name="Order_Constraint1" msdata:ConstraintName="Constraint1"
msdata:PrimaryKey="true">
      <xs:selector xpath=".//Order" />
      <xs:field xpath="Id" />
    </xs:unique>
    <xs:keyref name="Building_BuildingForCategory" refer="Building_Constraint1">
      <xs:selector xpath=".//BuildingForCategory" />
      <xs:field xpath="BuildingId" />
    </xs:keyref>
    <xs:keyref name="Category_BuildingForCategory"
refer="Category_Constraint1">
```

```
      <xs:selector xpath=".//BuildingForCategory" />
      <xs:field xpath="CategoryId" />
    </xs:keyref>
    <xs:keyref name="Category_Order" refer="Category_Constraint1">
      <xs:selector xpath=".//Order" />
      <xs:field xpath="CategoryId" />
    </xs:keyref>
    <xs:keyref name="Category_Exhibition" refer="Category_Constraint1">
      <xs:selector xpath=".//Exhibition" />
      <xs:field xpath="CategoryId" />
    </xs:keyref>
    <xs:keyref name="Order_Exposition" refer="Order_Constraint1">
      <xs:selector xpath=".//Exposition" />
      <xs:field xpath="OrderId" />
    </xs:keyref>
    <xs:keyref name="Building_Exposition" refer="Building_Constraint1">
      <xs:selector xpath=".//Exposition" />
      <xs:field xpath="BuildingId" />
    </xs:keyref>
    <xs:keyref name="Exhibition_Exposition" refer="Exhibition_Constraint1">
      <xs:selector xpath=".//Exposition" />
      <xs:field xpath="ExhibitionId" />
    </xs:keyref>
  </xs:element>
</xs:schema>
```

ДОДАТОК Е

Інтерфейс програми

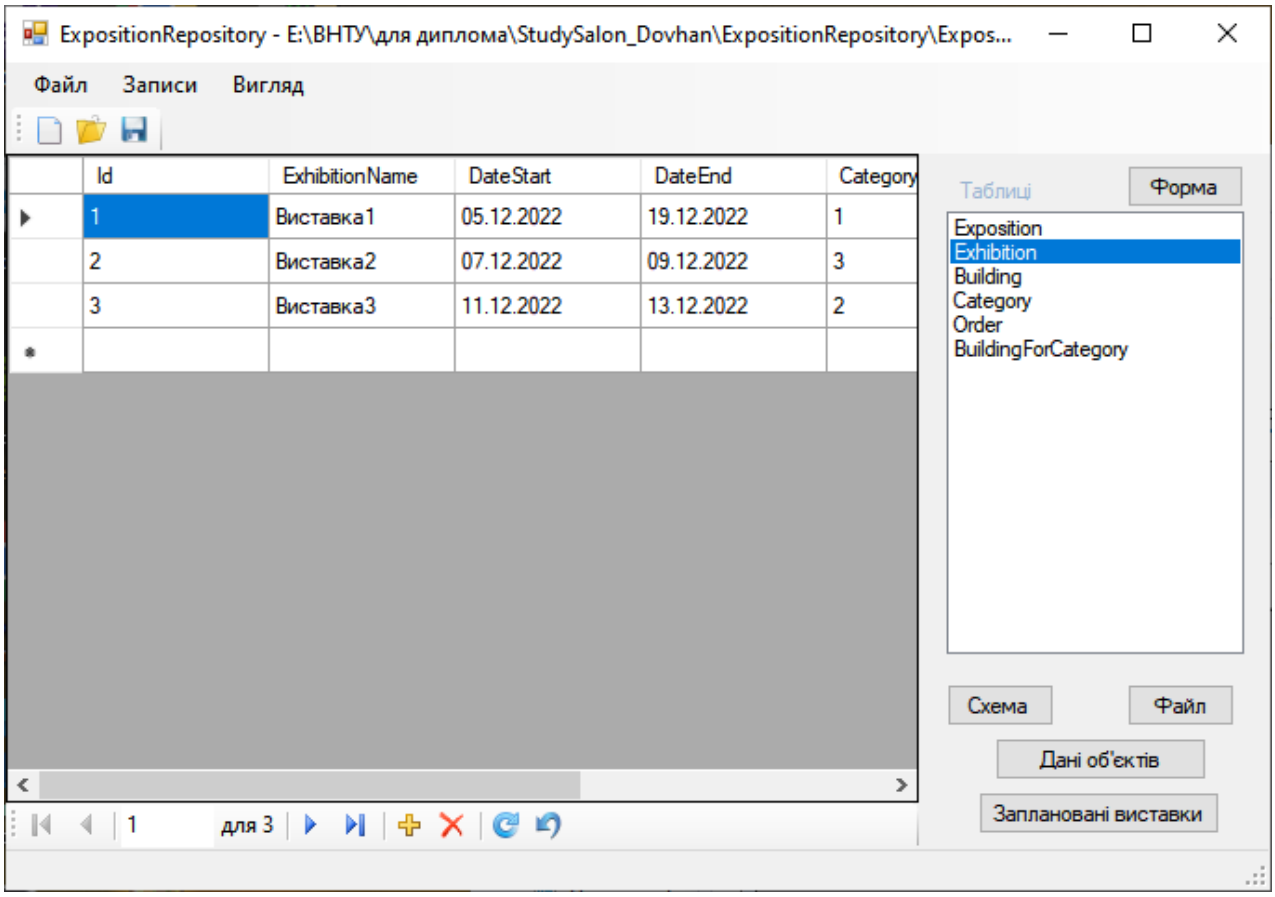

Рисунок Е.1 — Інтерфейс програми

ДОДАТОК Ж

Протокол перевірки дипломної роботи на наявність запозначень

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Підсистема зберігання даних обліку виставкового салону з вибірковим відновленням

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 90,5% Схожість 9,5%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.
- ☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- ☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Довгань О.В.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Черняк О.І.
(підпис) (прізвище, ініціали)