МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт з дисципліни
«ОСНОВИ ПРОГРАМУВАННЯ НА DELPHI»
для здобувачів першого (бакалаврського) рівня вищої освіти
спеціальності 123 (F7) Комп’ютерна інженерія
усіх форм навчання

2026

Методичні вказівки до виконання лабораторних робіт з дисципліни «Основи програмування на Delphi» для здобувачів першого (бакалаврського) рівня вищої освіти спеціальності 123 (F7) Комп’ютерна інженерія усіх форм навчання. / Укл. С.О. Сгадов – Запоріжжя: НУ «Запорізька політехніка», 2026. – 76 с.

Укладачі : С.О. Сгадов, ст. викладач

Рецензент : С.Ю. Скрупський, доцент, к.т.н.

Відповідальний за випуск С.О. Сгадов, ст. викладач

Затверджено:
На засіданні кафедри
«Комп’ютерні системи та мережі»
Протокол № 10 від 25 травня 2026 р.

Рекомендовано до видання
НМК факультету КНТ
Протокол № 11
від 09 червня 2026 р.

ЗМІСТ

С.

Вступ 5
1 Лабораторна робота № 1 – створення консольних програм 6
1.1 Теоретичні відомості 6
1.2 Порядок виконання лабораторної роботи 12
1.3 Варіанти індивідуальних завдань 12
1.4 Контрольні питання 14
2 Лабораторна робота № 2 – масиви 15
2.1 Теоретичні відомості 15
2.2 Порядок виконання лабораторної роботи 16
2.3 Варіанти індивідуальних завдань 17
2.4 Контрольні питання 19
3 Лабораторна робота № 3 – модулі, dll, файли 20
3.1 Теоретичні відомості 20
3.2 Порядок виконання лабораторної роботи 24
3.3 Варіанти індивідуальних завдань 25
3.4 Контрольні питання 26
4 Лабораторна робота № 4 – класи та об'єкти, класи-контейнери 27
4.1 Теоретичні відомості 27
4.2 Порядок виконання лабораторної роботи 30
4.3 Варіанти індивідуальних завдань 31
4.4 Контрольні питання 32
5 Лабораторна робота № 5 – властивості та методи компонентів у Delphi 33
5.1 Теоретичні відомості 33
5.2 Порядок виконання лабораторної роботи 44
5.3 Варіанти індивідуальних завдань 44
5.4 Контрольні питання 47

6 Лабораторна Робота № 6 – створення SDI-застосунку 48
6.1 Теоретичні відомості 48
6.2 Порядок виконання лабораторної роботи 50
6.3 Варіанти індивідуальних завдань 52
6.4 Контрольні питання 53
7 Лабораторна робота № 7 – створення MDI-застосунку 55
7.1 Теоретичні відомості 55
7.2 Порядок виконання лабораторної роботи 58
7.3 Варіанти індивідуальних завдань 63
7.4 Контрольні питання 63
8 Лабораторна робота № 8 – компоненти роботи з базами даних 64
8.1 Теоретичні відомості 64
8.2 Порядок виконання лабораторної роботи 70
8.3 Варіанти індивідуальних завдань 70
8.4 Контрольні питання 74
Перелік джерел посилання 76

ВСТУП

При виконанні лабораторного практикуму студент зобов'язаний:

- ознайомитися зі змістом майбутньої лабораторної роботи;

- повторити матеріал відповідного розділу курсу «Основи програмування на Delphi» та підготувати форму звіту (заповнити титульний аркуш, описати алгоритм програми та підготувати відповіді на контрольні запитання);

- виконати роботу та відповісти викладачеві на питання з даного розділу;

- у процесі самостійної підготовки оформити звіт, за потреби додати лістинг програми та сформулювати висновки щодо результатів;

- здати викладачеві повністю підготовлений та коректно оформлений звіт.

Звіт є коректно оформленим документом, що містить результати виконання лабораторного завдання. Звіт, підписаний студентом, надається викладачеві наприкінці лабораторного заняття. На титульному аркуші вказують номер і назву роботи, прізвище студента та номер групи.

Звіт повинен містити: формулювання завдання; короткий опис алгоритму; програмну реалізацію (мовою Object Pascal у середовищі Delphi); відповіді на контрольні запитання. Готовий звіт подається викладачеві наприкінці заняття.

Під час виконання лабораторних робіт кожен студент має безпосередньо брати участь у програмуванні, оформлювати та здавати звіт. Працюючи в лабораторії, необхідно дотримуватися правил техніки безпеки, установлених кафедрою.

1 ЛАБОРАТОРНА РОБОТА № 1 – СТВОРЕННЯ КОНСОЛЬНИХ ПРОГРАМ

Мета роботи: Навчитися використовувати RAD Studio для створення консольних застосунків.

1.1 Теоретичні відомості

Інтегроване середовище розробки IDE (від англ. *Integrated Development Environment*) – це комплекс програмних засобів, які розроблені для створення зручного робочого середовища, що реалізує концепцію швидкої розробки застосунків (RAD, *Rapid Application Development*). Воно слугує засобом взаємодії розробника з комп'ютером. За допомогою інструментів IDE програміст може проєктувати інтерфейсну частину програми, писати код, пов'язувати його з візуальними компонентами, проводити налагодження та тестовий запуск. Усе це відбувається в єдиному інтегрованому середовищі.

Середовище RAD Studio складається з кількох основних модулів, зокрема графічного інтерфейсу користувача, який умовно називають робочим столом (за аналогією з робочим столом ОС Windows). Він відкривається одразу після запуску Delphi.

Для початку роботи двічі клацніть піктограму RAD Studio на робочому столі або в меню «Пуск». Спочатку з'явиться заставка, що повідомляє про завантаження середовища в оперативну пам'ять.

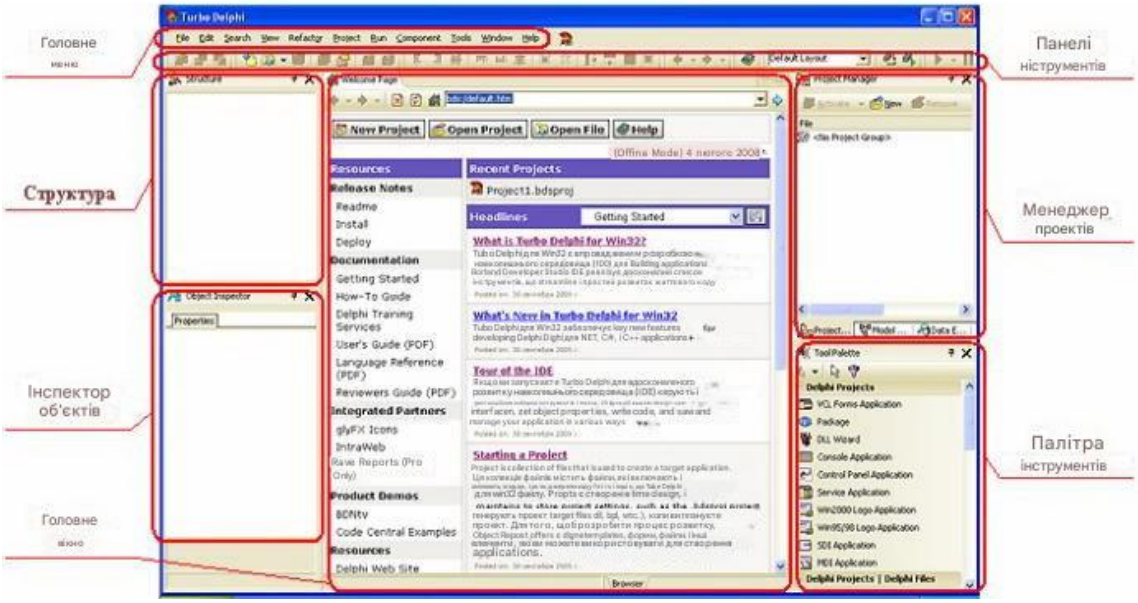

Рисунок 1.1 – Загальний вигляд RAD Studio

Після завантаження з'явиться робочий стіл RAD Studio з відображенням у головному вікні Welcome Page. Цю сторінку, як і всі інші, можна закрити, клацнувши правою кнопкою миші на корінці вгорі вікна та обравши команду Close Page. Назви додаткових панелей меню можна побачити, клацнувши правою кнопкою миші на полі, де вони розміщені.

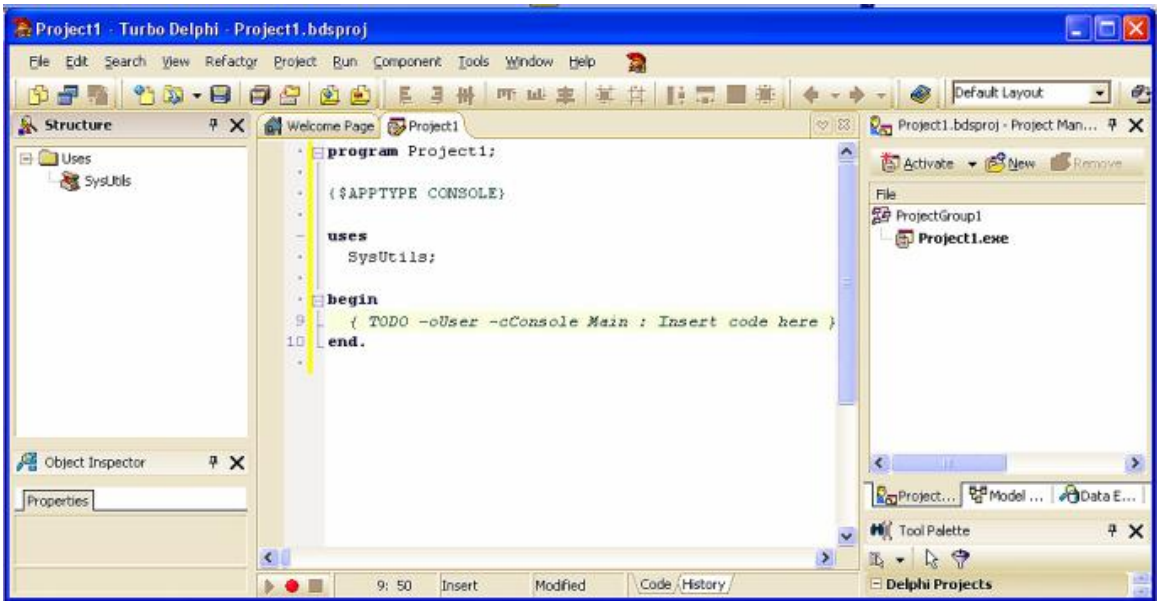

Рисунок 1.2 – Шаблон консольного застосунку.

Зліва та справа від головного вікна розташовані вікна Structure (Структура), Object Inspector (Інспектор об'єктів), Project Manager (Менеджер проєктів) та Tool Palette (Палітра інструментів).

Головне вікно обмежує простір, у якому відбувається розробка проєкту, створюється графічний інтерфейс користувача та робочий код програми. У нього розміщуються сторінки з модулями, які зазвичай складаються з трьох вкладок: Code, Design та History. Вкладка Design містить графічне зображення форми, а вкладка History – списки версій файлів, що входять до проєкту, а також змін до них. Вкладку Code зазвичай називають редактором коду (рис. 1.2 та 1.3).

Редактор коду є повнофункціональним текстовим редактором, за допомогою якого можна переглядати та редагувати вихідний код програми. Крім того, він містить численні засоби, що полегшують створення коду мовою Delphi.

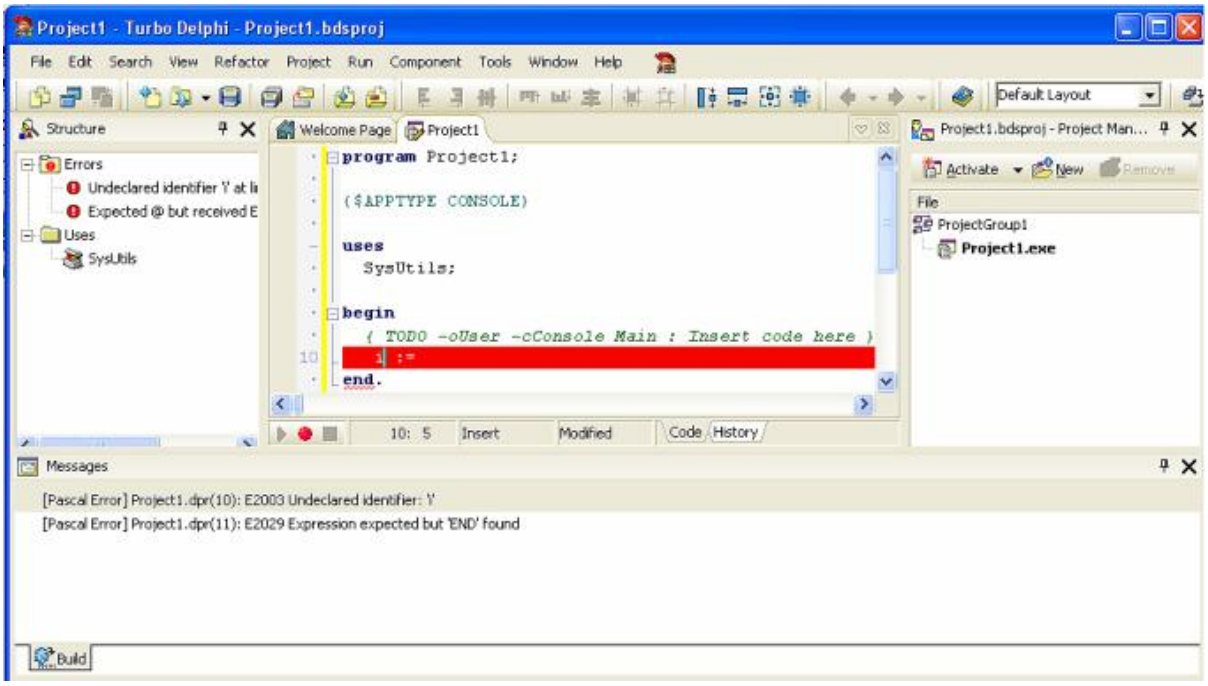

Рисунок 1.3 – Редактор коду

Разом з редактором коду зручно використовувати вікно Structure, за допомогою якого програміст може легко переглядати файли модулів. На

деревоподібній діаграмі показані всі типи, класи, властивості, методи, глобальні змінні та глобальні процедури, визначені у модулі, з яким відбувається робота, підключені модулі, виявлені синтаксичні помилки.

За замовчуванням біля вікна редактора коду відкриється вікно повідомлень Messages. Воно генерується автоматично, якщо під час компіляції програми було згенеровано повідомлення про помилки або попереджувальні повідомлення. Для відображення вікна повідомлень можна також клацнути правою кнопкою миші на полі редактора коду та у контекстному меню вибрати пункт Message View. Якщо двічі клацнути на повідомленні, у редакторі коду підсвічується відповідний рядок.

У недавньому минулому консоллю називали пристрій введення-виведення, призначений для зв'язку оператора обчислювальної системи з її керуючою програмою та завданнями. Діалог на консолі здійснювався в текстовому режимі за допомогою клавіатури та монітора. При запуску консольної програми Windows виділяє вікно, подібне до DOS-програми, проте вона є 32-розрядним застосунком Windows і не працює під MS-DOS. Виведення даних у консольних застосунках здійснюється за допомогою стандартних процедур write, writeln, а введення – read або readln, які можна використовувати без їх явного опису.

Для створення консольної програми необхідно обрати тип об'єкта (Delphi Projects → Console Application) у діалоговому вікні New Items, доступ до якого можна отримати через кнопку New Project або команду головного меню File → New → Other (див. рис. 1.4).

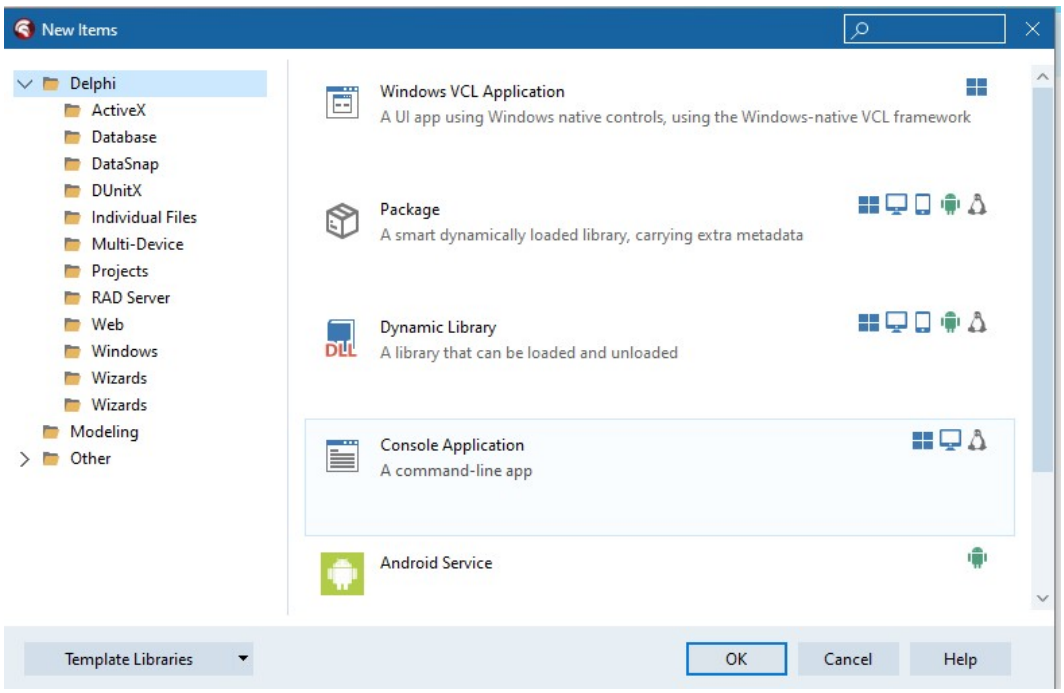

Рисунок 1.4 – Створення нового проєкта

Автоматично створений новий проєкт і з'явиться сторінка Project1 з шаблоном головної процедури консольної програми.

Лістинг 1.1 – Шаблон консольної програми.

```
program Project1;
{$APPTYPE CONSOLE}
uses
SysUtils ;
begin
{ TODO - oUser - cConsole Main : Insert code here }
end.
```

До шаблону консольної програми додаються необхідні оператори.

Слід звернути увагу, що консольний застосунок створюється в Windows, а виконується з використанням відмінного від Windows кодування символів. Проблему перекодування та коректного виведення повідомлень українською мовою легко вирішити, вказавши число 866 в опції компілятора Codepage як

номер кодової сторінки національного алфавіту за допомогою команди меню **Project → Options → Compiling** (рис. 1.5)**.**

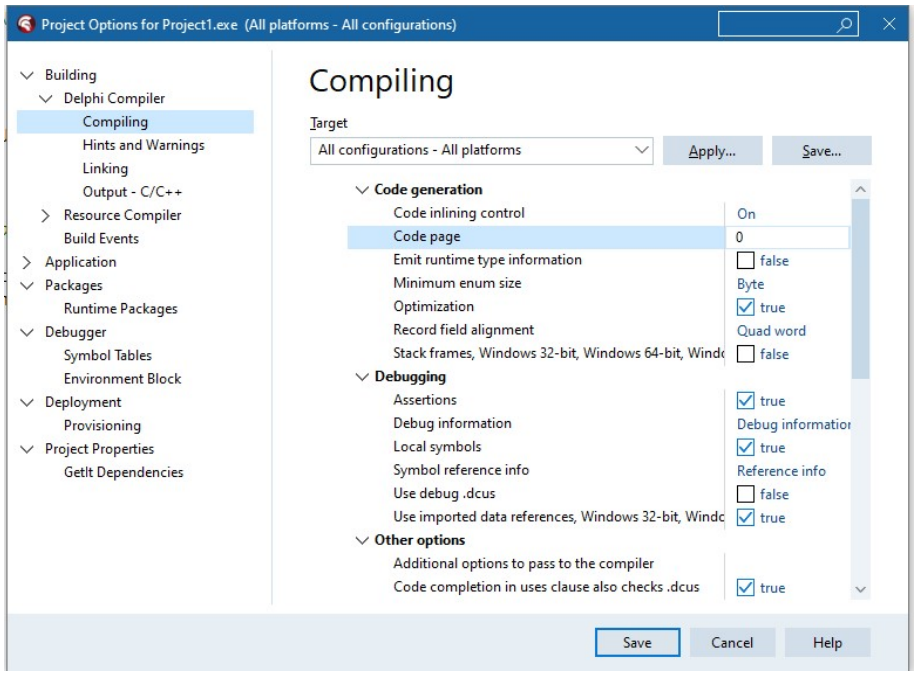

Рисунок 1.5 – Опції компілятора.

Консольний застосунок RAD Studio є не просто програмою, написаною мовою Object Pascal, яка виконується у середовищі Windows. RAD Studio підтримує створення 32-розрядних консольних застосунків, які мають доступ до ресурсів системи та використовують функції API Windows. При цьому в розділі uses необхідно підключати модулі, ресурси яких використовуються у програмі (лістинг 1.2).

Лістинг 1.2 – Приклад використання в консольній програмі функцій API

```
program Console; {$APPTYPE CONSOLE}
 uses SysUtils, Windows ;
Var Code_Page : Integer;
begin
     SetConsoleTitle ('Консольний застосунок');
     Code_Page := GetConsoleOutputCP ;
     WriteLn (' Кодова сторінка = ', Code_Page : 3);readln ;
end.
```

Програма змінює заголовок вікна за допомогою API-функції `SetConsoleTitle`. Функція `GetConsoleOutputCP` повертає номер кодової сторінки національного алфавіту. Оскільки ці функції входять до Windows API, ім'я модуля `Windows` потрібно вказати в розділі `uses`.

1.2 Порядок виконання лабораторної роботи

1 Створіть консольний застосунок згідно з варіантом.

2 Скомпілюйте та запустіть. Зробіть скриншоти.

1.3 Варіанти індивідуальних завдань

1. Дано два цілих числа A і B (A < B). Вивести всі цілі числа між ними (включно), у порядку зростання, а також кількість N.

2. Дано два цілих числа A і B (A < B). Вивести всі цілі числа між ними (не включаючи), у порядку спадання, а також кількість N.

3. Дано дійсне число A та ціле число N (> 0). Вивести A у степені N.

4. Дано дійсне число A та ціле число N (> 0). Вивести всі цілі степені числа від 1 до N.

5. Дано дійсне число A та ціле число N (> 0). Вивести $1+A+A^2+A^3+...+A^N$.

6. Дано дійсне число A і ціле число N (> 0). Вивести $1 – A + A^2 – A^3 + ... + (–1)^N A^N$.

7. Дано ціле число N (>1). Вивести найменше ціле K, при якому $3^K > N$, і саме значення 3^K.

8. Дано ціле число N (> 1). Вивести найбільше ціле K, при якому $3^K < N$, і саме значення 3^K.

9. Дано дійсне число A (> 1). Вивести найменше ціле N, для яких сума 1 + 1/2 + ... + 1/N буде більшою за A, і саму цю суму.

10. Дано дійсне число A (> 1). Вивести найбільше ціле N, для яких сума 1 + 1/2 + ... + 1/N буде меншою за A, і саму цю суму.

11. Дано ціле число N (> 0). Вивести добуток 1·2·...·N. Щоб уникнути переповнення цілого типу, обчислювати цей добуток за допомогою дійсної змінної та виводити його як дійсне число.

12. Дано ціле число N(>0). Якщо N – непарне, вивести добуток 1·3·...·N; якщо парне, то 2·4·...·N. Щоб уникнути переповнення цілого типу, обчислювати за допомогою дійсної змінної.

13. Дано ціле число N(>0). Вивести суму 2 + 1/(2!) + 1/(3!) + ... + 1/(N!). Отримане число є наближеним значенням константи e.

14. Дано дійсне число X та ціле число N (> 0). Вивести $1+X+X^2/2! + ... + X^N/N!$ (наближене значення функції exp у точці X).

15. Дано дійсне число X та ціле число N (> 0). Вивести $X – X^3/3! + X^5/5! – … + (–1)^N X^{2N+1}/(2N+1)!$ (наближене значення функції sin).

16. Дано дійсне число X та ціле число N (> 0). Вивести $1-X^2/2!+X^4/4!–…+(–1)^N X^{2N}/(2N)!$ (наближене значення функції cos).

17. Дано дійсне число X (|X| < 1) і ціле число N (> 0). Вивести $X – X^2/2 + X^3/3 – ... + (–1)^{N–1} X^N/N$ (наближене значення функції ln у точці 1+X).

18. Дано дійсне число X (|X| < 1) і ціле число N (> 0). Вивести $X – X^3/3 + X^5/5 – ... + (–1)^N X^{2N+1}/(2N+1)$ (наближене значення функції arctg).

19. Дано ціле число N (> 2) та дві дійсні точки на числовій осі: A, B (A<B). Відрізок [A, B] розбитий на рівні відрізки довжини H. Вивести значення H і набір N точок.

20. Дано ціле число N (> 2) та дві дійсні точки A, B (A < B). Функція F(X) = 1 – sin(X). Вивести значення функції F в N рівновіддалених точках, що утворюють розбиття відрізка [A, B].

21. Дано число D>0. Послідовність AN: $A_1 = 2$, $A_n = 2 + 1/A_{n-1}$. Знайти перший номер K, для якого $|A_k - A_{k-1}| < D$, і вивести цей номер, а також числа A_{k-1} та A_k.

22. Дано число D>0. Послідовність AN: $A_1 = 1$, $A_2 = 2$, $A_n = (A_{n-2} + A_{n-1})/2$. Знайти перший номер K, для якого $|A_k - A_{k-1}| < D$, і вивести цей номер, а також числа A_{k-1} та A_k.

1.4 Контрольні питання

1. Назвіть основні файли, які входять до складу проєкту.
2. Які файли (з якими розширеннями) з'являються в робочій теці після компіляції проєкту?
3. Перелічіть основні розділи програмного коду модуля.
4. Яка інформація відображається у вікні Project Manager?
5. Що буде, якщо в ході перенесення з машини на машину загубиться файл із розширенням dcu?

2 ЛАБОРАТОРНА РОБОТА № 2 – МАСИВИ

Мета роботи: навчитися працювати з масивами та процедурами

2.1 Теоретичні відомості

2.1.1 Масиви

Робота з масивами є фундаментальною частиною алгоритмічного мислення та структурованого програмування. У середовищі Delphi підтримуються різні підходи до оголошення та маніпулювання масивами.

Починаючи з версії Delphi XE, компанія Embarcadero розширила синтаксичні можливості мови, дозволивши ініціалізувати динамічні масиви безпосередньо під час оголошення: `dynArray: array of integer = [0, 1, 2, 3];`.

Статичні масиви характеризуються фіксованою довжиною, яка визначається на етапі компіляції: `array [Lower..Upper] of Type`. Вони розташовуються переважно в стеку або сегменті даних, що робить доступ до них швидким.

Динамічні масиви виділяються в динамічній області пам'яті та можуть змінювати розмір під час виконання програми за допомогою процедури `SetLength`. Сучасний синтаксис дозволяє очищати їх присвоєнням порожнього літералу: `dynArray := [];`.

Для багатовимірних структур доцільно використовувати узагальнений тип `TArray<T>`, який надає типобезпечний інтерфейс. Наприклад, двовимірний масив можна оголосити як `TArray<TArray<Integer>>`.

Для динамічних масивів доступні операції конкатенації, вставки (`Insert`), видалення (`Delete`) та копіювання (`Copy`). Для зміни розміру

використовується `SetLength`, а для отримання інформації про розмірність – функції `DynArrayBounds` та `DynArrayClear`.

2.1.2 Процедури та функції: сучасні можливості

Синтаксис виклику процедур і функцій без параметрів допускає два еквівалентні варіанти: `Form1.Show;` та `Form1.Show();`.

За замовчуванням параметри передаються за значенням. Якщо підпрограма повинна модифікувати передані дані, використовують ключове слово `var`. Якщо дані не змінюються, найкращою практикою є оголошення параметра-константи:

```
procedure AnalyzeMatrix(const M: TArray<TArray<Double>>);.
```

Перевантаження підпрограм реалізується за допомогою директиви `overload`. Параметри за замовчуванням оголошуються після знака рівності та обов'язково розташовуються в кінці списку. Механізм відкритих масивів дозволяє передавати довільну кількість однотипних аргументів: `function CalculateSum(const Values: array of Double): Double;`.

Повернення результату здійснюється через неявну змінну `Result`. Директива `{$X+}` (увімкнена за замовчуванням) дозволяє ігнорувати повернене значення функції.

2.2 Порядок виконання лабораторної роботи

Виконання роботи передбачає створення консольного застосунка відповідно до індивідуального варіанту, реалізацію необхідної функції або процедури безпосередньо у файлі проєкту з розширенням `.dpr`, компіляцію та

запуск програми з подальшим фіксуванням результатів за допомогою скриншотів.

1. Створіть консольний застосунок за варіантом.

2. Реалізуйте локальну функцію або процедуру за варіантом у файлі `*.dpr` та використайте її.

3. Скомпілюйте та запустіть.

4. Зробіть скриншоти, підпишіть їх та помістіть у звіт.

2.3 Варіанти індивідуальних завдань

1. Описати функцію `Polynom(A, N, X)` дійсного типу, що знаходить значення полінома P у дійсній точці X. Поліном P визначається параметрами N (ступінь, 0 < N < 8) і A (коефіцієнти, дійсний масив розміру N+1).

2. Описати функцію `Min(A, N)` дійсного типу, що знаходить мінімальний елемент масиву A.

3. Описати функцію `Max(A, N)` дійсного типу, що знаходить максимальний елемент масиву A.

4. Описати функцію `NMin(A, N)` цілого типу, що знаходить номер мінімального елемента масиву A.

5. Описати функцію `NMax(A, N)` цілого типу, що знаходить номер максимального елемента масиву A.

6. Описати процедуру `NMinmax(A, N, NMin, NMax)`, що повертає номери мінімального та максимального елементів масиву A.

7. Описати функцію `NODN(A, N)` цілого типу, що знаходить найбільший спільний дільник елементів масиву цілих чисел A.

8. Описати функцію `NOKN(A, N)` цілого типу, що знаходить найменше спільне кратне елементів цілочислового масиву A.

9. Описати процедуру `Factors(A, N, F)`, яка розкладає натуральне число A на прості множники. Кількість множників повертається в N, самі множники (у порядку неспадання) – у цілочисловому масиві F.

10. Описати функцію `PerimN(X, Y, N)` дійсного типу, що повертає периметр N-кутника за координатами вершин у масивах X, Y.

11. Описати процедуру `Invert(A, N)`, що змінює порядок елементів масиву A на протилежний.

12. Описати процедури `MoveLeft(A, N, k)` та `MoveRight(A, N, k)` для циклічного зсуву дійсного масиву.

13. Описати процедуру `Smooth(A, N)` для згладжування масиву середнім арифметичним сусідів.

14. Описати процедуру `RemoveX(A, N, X)` для видалення елементів, рівних X, з масиву цілих чисел.

15. Описати процедуру `DoubleX(A, N, X)` для дублювання елементів, рівних X.

16. Описати процедури `SortInc(A, N)` та `SortDec(A, N)` для сортування масиву за зростанням або спаданням.

17. Описати функцію `Norm1(A, M, N)` дійсного типу для обчислення норми матриці за стовпцями.

18. Описати функцію `Norm2(A, M, N)` дійсного типу для обчислення норми матриці за рядками.

19. Описати функцію `SumLine(A, M, N, k)` для обчислення суми елементів k-го рядка матриці.

20. Описати процедуру `SwapLine(A, M, N, k1, k2)` та `SwapCol(A, M, N, k1, k2)` для обміну рядків/стовпців матриці.

21. Описати процедуру `Transp(A, M)` для транспонування квадратної матриці.

22. Описати процедуру `Gauss(A, M, N, i1, i2, X)` для елементарних перетворень рядків матриці.

23. Описати процедуру `DelIJ(A, M, N, i, j)` для видалення рядка та стовпця, що містять елемент A[i,j].

2.4 Контрольні питання

1. Як оголосити масив у Delphi?
2. Як оголосити та використовувати динамічний масив у Delphi?
3. Як отримати розмір динамічного масиву?
4. Як встановити параметри процедури?
5. Які види параметрів процедури бувають?
6. Як передати масив у процедуру?

3 ЛАБОРАТОРНА РОБОТА № 3 – МОДУЛІ, DLL, ФАЙЛИ

Мета роботи: навчитися писати та використовувати модулі, DLL, навчитися працювати з файлами.

3.1 Теоретичні відомості

3.1.1 Робота з файлами

Під файлом розуміється або іменована область зовнішньої пам'яті ПК, або логічний пристрій – потенційне джерело або приймач інформації. Файл є сукупністю компонентів одного типу. Типом компонентів може бути будь-який тип Pascal, крім файлів. Файловий тип або змінну файлового типу можна задати одним із трьох способів:

```
<ім'я> = FILE OF <тип>;
<ім'я> = TEXT;
<ім'я> = FILE;
```

Тут <ім'я> – ідентифікатор файлового типу; TEXT – стандартний тип текстових файлів; <тип> – будь-який тип даних, крім файлів. Залежно від способу оголошення виділяються три види файлів: типізовані файли (FILE OF ...), текстові файли (TEXT або TEXTFILE) та нетипізовані файли (FILE).

Для доступу до файлів використовується файлова змінна, яка пов'язується з фізичним файлом процедурою AssignFile. Відкриття наявного файлу здійснюється процедурою Reset, створення та відкриття нового – Rewrite. Після виконання операцій файл закривається процедурою CloseFile.

При роботі з файлами можливі помилки вводу-виводу. Їх можна обробляти двома шляхами: через виключні ситуації EInOutError або за

допомогою опції {$I-} з подальшою перевіркою функції IOResult. Перевірка закінчення файлу виконується функцією Eof.

Текстові файли складаються з послідовностей символів, розбитих на рядки. Запис даних здійснюється процедурами Write та Writeln, читання – Read та Readln. У Delphi роботу з текстовими файлами також можна організувати через методи LoadFromFile та SaveToFile класів TStrings і TStringList, що інтегровані у компоненти TMemo та TRichEdit.

Типізовані файли є двійковими файлами, що містять послідовність однотипних даних. Процедури читання та запису Read та Write вимагають точної відповідності типів. Для довільного доступу використовується процедура Seek, а поточну позицію повертає функція FilePos.

Нетипізовані файли містять послідовність байтів. Відкриття здійснюється процедурами Reset та Rewrite з додатковим параметром RecSize (розмір запису в байтах). Читання та запис виконуються процедурами BlockRead та BlockWrite, що працюють з блоками пам'яті.

Дескриптори файлів – низькорівневий API Windows. Функції FileOpen, FileCreate, FileRead, FileWrite та FileClose працюють із цілочисельними дескрипторами. Цей метод доцільний лише для спеціалізованих завдань.

3.1.2 Модулі

Модуль (unit) – це окремий вихідний файл .pas, що містить логічно пов'язані оголошення типів, констант, змінних, процедур та функцій. Структура модуля включає заголовок, розділи interface, implementation, а також необов'язкові секції initialization та finalization.

Розділ interface містить публічні оголошення, доступні іншим модулям через директиву uses. Тут описуються сигнатури процедур, функцій, класів та

змінних. Реалізація цих оголошень розміщується у розділі implementation, який може містити також приватні допоміжні об'єкти, недоступні ззовні.

Директива uses може з'являтися як в interface, так і в implementation. Ідентифікатори, оголошені в модулях, перелічених в implementation, доступні лише всередині цього розділу. Секція initialization містить код, що виконується під час завантаження модуля. Секція finalization виконується при завершенні програми у зворотному порядку і слугує для звільнення ресурсів.

Імена модулів мають збігатися з назвами файлів та бути унікальними в межах проєкту. При компіляції кожен модуль створює окремий .dcu-файл, який потім компонується у виконуваний модуль.

3.1.3 Динамічні бібліотеки (DLL)

Основні сфери використання DLL: спільні бібліотеки функцій, ресурсні сховища (піктограми, малюнки, рядки), бібліотеки підтримки (DirectX, OpenGL тощо), частини програми (форми), плагіни для розширення функціоналу, а також спільне використання ресурсів кількома процесами.

Імпортування функцій із DLL

1. Статичне імпортування (прив'язка). Найпростіший метод. Функція оголошується з директивою external:

```
function FunctionName(Par1: Par1Type; Par2: Par2Type): ReturnType; stdcall;
external 'DLLNAME.DLL' name 'FunctionName';
```

Директива stdcall забезпечує сумісність угоди про виклик із Windows API. Недолік: якщо DLL відсутня, програма не запуститься.

2. Динамічне завантаження. Бібліотека завантажується під час виконання через LoadLibrary, адреса функції отримується через GetProcAddress, а звільнення пам'яті виконується FreeLibrary:

```
var
```

```
 LibHandle: THandle;
 GetSimpleText: function(LangRus: Boolean): PChar; stdcall;
begin
 @GetSimpleText := nil;
 LibHandle := LoadLibrary('MYDLL.DLL');
 if LibHandle >= 32 then
 begin
  @GetSimpleText := GetProcAddress(LibHandle, 'GetSimpleText');
  if @GetSimpleText <> nil then
   ShowMessage(StrPas(GetSimpleText(True)));
 end;
 FreeLibrary(LibHandle);
end;
```

При передачі рядків між програмою та DLL слід уникати типу string через різницю в керуванні пам'яттю. Рекомендується використовувати PChar з подальшим перетворенням через StrPas.

Експорт функцій із DLL виконується через директиву exports:

```
library mydll;
uses SysUtils, Classes;
function GetSimpleText(LangRus: Boolean): PChar; stdcall;
begin
 if LangRus then
  Result := PChar('Привіт, світ!')
 else
  Result := PChar('Hello, world!');
end;
exports GetSimpleText;
begin
end.
```

3.1.4 Розміщення форм та створення плагінів

Для розміщення модальної форми у DLL створюється функція, яка ініціалізує екземпляр форми, встановлює параметри, викликає ShowModal і звільняє пам'ять:

```
function ShowMyDialog(Msg: PChar): Boolean; stdcall;
var
 Form1: TForm1;
begin
 Form1 := TForm1.Create(Application);
 try
  Form1.Label1.Caption := StrPas(Msg);
  Result := (Form1.ShowModal = mrOk);
 finally
  Form1.Free;
 end;
end;
```

Створення плагінів базується на стандартному інтерфейсі: функція → оголошення (наприклад, GetPluginName) → основна функція обробки даних. Головна програма динамічно завантажує DLL, викликає ідентифікаційну функцію для побудови меню, а при виборі – передає дані в основну функцію плагіна.

3.2 Порядок виконання лабораторної роботи

1. Створіть модуль (Unit), який містить (експортує) вказані у варіанті функції. Реалізуйте програму, яка використовує ці функції. Оскільки проєкт міститиме більше одного файлу, дотримуйтесь правила: один проєкт – один

каталог. Усі вхідні рядкові параметри, що задають імена файлів, доцільно описувати як параметри-константи.

2. На базі функцій, написаних у попередньому пункті, створіть DLL, яка експортує ці функції. Реалізуйте дві програми-клієнти: одна зі статичним імпортом (через external), інша – з динамічним завантаженням (LoadLibrary / GetProcAddress). Заголовки функцій для статичного імпорту винесіть в окремий модуль. Не забудьте вивантажити динамічно завантажену DLL перед закінченням програми.

3.3 Варіанти індивідуальних завдань

1. Описати функцію getInt(Name, k) цілого типу, що повертає k-й елемент файлу цілих чисел з ім'ям Name (елементи нумеруються від 0). Якщо файл не існує або не містить k-го елемента, функція повертає 0. Вивести п'ять елементів даного файлу із зазначеними номерами.

2. Описати функцію getLine(Name, k) рядкового типу, що повертає k-й рядок текстового файлу з ім'ям Name (рядки нумеруються від 0). Якщо файл не існує або не містить рядка, функція повертає порожній рядок. Вивести п'ять рядків файлу з вказаними номерами.

3. Описати функцію IntFileSize(Name) цілого типу, що повертає розмір файлу цілих чисел з ім'ям Name. Якщо файл не існує, функція повертає -1. Визначити розмір трьох файлів із даними іменами.

4. Описати функцію TextSize(Name) цілого типу, що повертає кількість рядків у текстовому файлі з ім'ям Name. Якщо файл не існує, функція повертає –1. Визначити розмір трьох файлів із даними іменами.

5. Описати процедуру InvertIntFile(Name), що змінює порядок проходження елементів файлу цілого типу з ім'ям Name на протилежний. Якщо

файл не існує або містить менш як два елементи, процедура не виконує жодних дій. Обробити три файли з цими іменами.

6. Описати процедуру SplitIntFile(Name0, k, Name1, Name2), що копіює перші k елементів файлу цілих чисел з ім'ям Name0 у файл Name1, а решту елементів – у файл Name2. Застосувати процедуру до файлу Name0.

7. Описати процедуру SplitText(Name0, k, Name1, Name2), що копіює перші k рядків текстового файлу з ім'ям Name0 у файл Name1, а решту – у файл Name2. Застосувати процедуру до файлу Name0.

8. Описати процедуру ConcatFile(NameA, NameB, NameAB), що об'єднує вміст двох двійкових файлів NameA та NameB одного типу у новому файлі NameAB. Використовувати BlockRead та BlockWrite. Застосувати до пар файлів.

9. Описати процедуру CodeText(Name, k), що шифрує текстовий файл з ім'ям Name, виконуючи циклічну заміну кожної літери на букву, розташовану в алфавіті на k-й позиції після вихідної ($0 < k < 11$). Літеру "ґ" не враховувати. Зашифрувати та розшифрувати файл із відомим k.

3.4 Контрольні питання

1. Види файлів у Pascal.
2. Яким є порядок дій, щоб відкрити файл?
3. Як ввести або вивести щось у файл?
4. Як помістити та експортувати функцію/процедуру в модуль?

4 ЛАБОРАТОРНА РОБОТА № 4 – КЛАСИ ТА ОБ'ЄКТИ, КЛАСИ-КОНТЕЙНЕРИ

Мета роботи: опанувати основи об'єктно-орієнтованого програмування в Delphi, навчитися створювати власні класи, використовувати механізми спадкування та властивостей, а також застосовувати стандартні класи-контейнери TList, TStrings та TStringList для організації колекцій даних.

4.1 Теоретичні відомості

4.1.1 Основи класів у Delphi

Клас у Delphi є спеціальним типом даних, що структурно нагадує запис, але оголошується за допомогою службового слова class. Він об'єднує поля для зберігання стану, методи для обробки даних та властивості для контрольованого доступу до внутрішньої структури. Інкапсуляція реалізується шляхом обмеження видимості полів та методів у розділах private, protected та public. Змінна типу клас завжди є посиланням на динамічно виділену пам'ять, тому оператор розіменування ^ не використовується. Екземпляр створюється конструктором Create та знищується методом Free або деструктором Destroy. Параметр Self неявно передається в кожен метод і вказує на поточний об'єкт, забезпечуючи доступ до його полів та інших методів без явного уточнення імені екземпляра.

4.1.2 Спадкування та перевизначення

Будь-який клас може породжувати похідний тип, який успадковує всі поля, методи та властивості базового класу. Якщо в нащадку оголошується метод з таким самим іменем, як у базового класу, відбувається його перевизначення. Для коректної роботи поліморфізму базовий метод має містити директиву virtual, а перевизначений у похідному класі – override. Доступ до реалізації базового класу забезпечується службовим словом inherited. Усі класи в Delphi неявно успадковуються від TObject, який надає базові методи керування життєвим циклом та інформацією про тип. При перевизначенні методів рекомендується дотримуватися ієрархії від простих сутностей до складних, послідовно розширюючи функціональність на кожному рівні спадкування.

4.1.3 Властивості (Properties)

Властивості регламентують доступ до полів класу, дозволяючи виконувати додаткові дії під час читання чи запису значень. Властивості оголошуються ключовим словом property та містять специфікатори read і write, які посилаються на поле або метод. Якщо специфікатор вказує на метод, read вимагає функції без параметрів, що повертає значення того ж типу, а write – процедури з одним параметром-значенням або const-параметром. Властивості не мають фізичної адреси в пам'яті, тому їх не можна передавати як var-параметри чи брати їх адресу оператором @. Властивість може бути доступна лише для читання або лише для запису, якщо вказано лише один зі специфікаторів.

4.1.4 Клас-контейнер TList

Клас TList реалізує динамічний індексований список нетипізованих вказівників Pointer, що дозволяє зберігати елементи довільних типів. Властивість Count відображає фактичну кількість елементів, а Capacity – поточну ємність виділеної пам'яті. Доступ до елементів здійснюється через властивість Items або оператор квадратних дужок. Методи Add, Insert, Delete та Clear керують структурою списку. Важливо враховувати, що TList не керує пам'яттю збережених об'єктів. При очищенні списку або видаленні елементів програміст повинен вручну викликати Free для кожного об'єкта, щоб уникнути витоків пам'яті. Метод Sort дозволяє впорядкувати список за допомогою користувацької функції порівняння, яка приймає два вказівники та повертає ціле число.

4.1.5 Класи TStrings та TStringList

TStrings є абстрактним базовим класом для роботи зі списками рядків і не використовується безпосередньо, оскільки містить абстрактні методи. Його практичний нащадок TStringList реалізує повноцінний список, що підтримує пари «рядок-об'єкт» та методи LoadFromFile і SaveToFile для роботи з текстовими файлами. Властивість Sorted вмикає автоматичне абеткове сортування при додаванні нових елементів, а Duplicates керує поведінкою при спробі додати однакові рядки. Метод CustomSort дозволяє застосовувати власну логіку порівняння через функцію, що приймає індекси рядків. Властивість CommaText об'єднує всі рядки в один рядок з роздільниками-комами, що зручно для серіалізації даних.

4.1.6 Графічні класи: TCanvas, TPen, TBrush, TFont

Візуалізація в Delphi базується на класі TCanvas, який представляє графічну поверхню (канву) компонентів. Властивість Pen визначає колір, стиль і товщину ліній, Brush відповідає за заливку замкнених фігур, а Font керує параметрами тексту. Координатна система канви починається з лівого верхнього кута (0, 0), а доступ до окремих пікселів забезпечує масив Pixels. Методи LineTo, MoveTo, Rectangle, Ellipse, Polygon та TextOut виконують базові операції малювання. Процедура FloodFill заповнює замкнену область заданим кольором або пензлем, а TextRect дозволяє виводити текст з обрізанням за межами прямокутника. Усі графічні операції мають виконуватися в межах подій OnPaint або після явного блокування канви методом Lock.

4.2 Порядок виконання лабораторної роботи

Створіть VCL- застосунок з однією формою та збережіть проєкт в окремій робочій теці. Оголосіть базовий клас TMyPoint, що містить поля координат X, Y, маси M, а також методи Create, Move та Draw. Створіть похідні класи для графічних об'єктів вашого варіанта, перевизначивши метод Draw для візуалізації на канві форми. Реалізуйте клас-контейнер TUniverse, що зберігає список об'єктів у TList або TObjectList. У конструкторі контейнера створіть екземпляри фігур, а в деструкторі забезпечте коректне звільнення пам'яті через цикл від Count-1 downto 0. У методі Draw контейнера організуйте перебір списку та виклик методу Draw для кожного об'єкта. Додайте обробник події OnPaint форми або таймер для анімації руху. Скомпілюйте та запустіть програму, переконавшись у відсутності витоків пам'яті. Зробіть скриншоти інтерфейсу, коду класів та результатів виконання.

4.3 Варіанти індивідуальних завдань

Створіть об'єктно-орієнтовану модель та візуалізуйте на формі екземпляри класів згідно з варіантом. Усі об'єкти мають бути похідними від базового класу TMyPoint та зберігатися у класі-контейнері.

1. 10 сніжинок TSnowflake різного кольору та машинка TCar. Реалізуйте анімацію падіння сніжинок та руху машинки.

2. 10 маленьких кіл TCircle різного кольору та розміру. Кола мають випадково рухатися у межах форми.

3. 5 кіл TCircle та 10 зірочок TStar різного кольору та розміру. Зірочки мерехтять (змінюють прозорість або колір).

4. 5 трикутників TTriangle та 5 зірочок TStar. Трикутники обертаються навколо власного центру.

5. 5 трикутників TTriangle та 5 кіл TCircle. Об'єкти відштовхуються при зіткненні (спрощена фізика).

6. 10 зірочок TStar та 3 машинки TCar. Машинки рухаються за заданою траєкторією, зірочки — випадково.

7. 5 зафарбованих квадратів TSquare та 10 зірочок TStar. Квадрати змінюють колір при натисканні миші.

8. 5 квадратів TSquare та 5 кіл TCircle. Об'єкти переміщуються клавішами навігації.

9. 10 сніжинок TSnowflake та машинка TCar. Сніжинки накопичуються внизу форми, машинка збирає їх.

10. 10 хмарок TCloud різного кольору та машинка TCar. Хмарки пливуть зліва направо, машинка рухається за мишею.

11. 10 хмарок TCloud та 5 зірочок TStar. Хмарки закривають зірочки, коли проходять повз них.

4.4 Контрольні питання

1. Що таке клас у Delphi та які основні елементи він містить?

2. У чому різниця між полем, методом і властивістю класу?

3. Як створюється та знищується екземпляр класу? Чому важливо використовувати try..finally?

4. Як реалізується спадкування та перевизначення методів? Для чого потрібні директиви virtual та override?

5. Як працює параметр Self і коли його варто використовувати явно?

6. Які обмеження існують при оголошенні властивостей (специфікатори read / write)?

7. Для чого використовується клас TList і які його основні методи?

8. Чому при видаленні об'єктів зі списку TList потрібно вручну викликати Free?

9. Як забезпечити автоматичне сортування та унікальність рядків у TStringList?

10. Які основні властивості та методи має клас TCanvas для малювання?

11. Як змінити колір, стиль та товщину лінії під час малювання на Canvas?

12. Як забезпечити коректне звільнення пам'яті при роботі з динамічними масивами об'єктів?

13. Як обробляти події миші та клавіатури для інтерактивного керування об'єктами на формі?

14. Які переваги дає використання TObjectList замість TList при роботі з об'єктами?

5 ЛАБОРАТОРНА РОБОТА № 5 – ВЛАСТИВОСТІ ТА МЕТОДИ КОМПОНЕНТІВ У DELPHI

Мета роботи: навчитися створювати віконні програми у середовищі RAD Studio

5.1 Теоретичні відомості

5.1.1 Порядок роботи з VCL

Для початку роботи з VCL створіть новий проєкт клацнувши мишею елемент головного меню **File**, у списку слід обрати **New** і в підменю обрати елемент **Windows VCL Application-Delphi**, тим самим буде створено порожню форму (рис. 5.1).

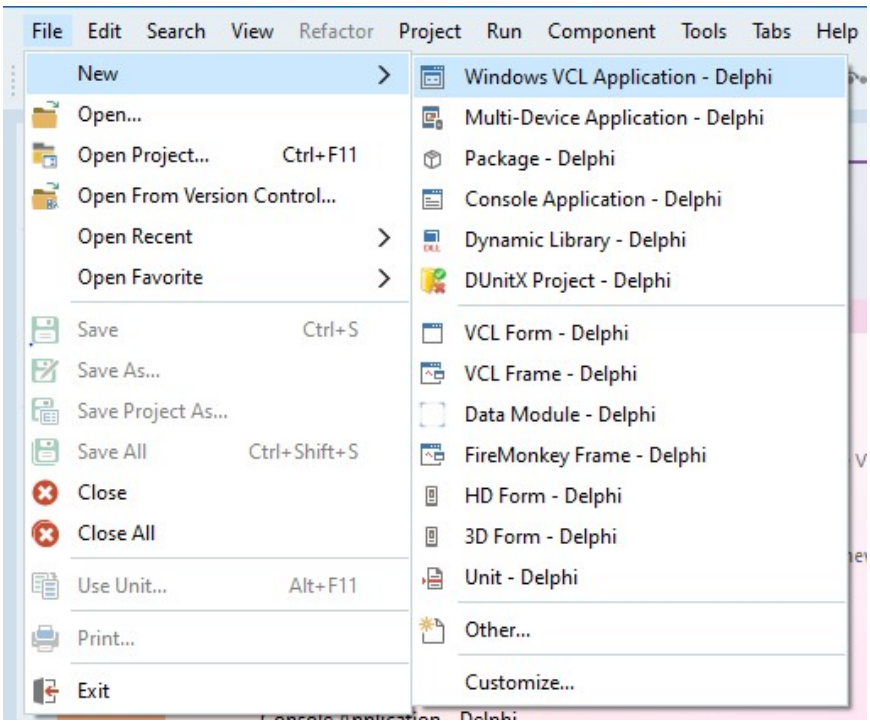

Рисунок 5.1 – Створення VCL застосунку

У центрі розташовується форма, у лівому нижньому куті – **Інспектор об'єктів** (Object Inspector), у правому нижньому куті – **Панель інструментів** (Tool Palette).

Інспектор об'єктів дозволяє змінювати властивості обраного на формі компонента. Він містить дві вкладки: Properties (властивості) та Events (події). У вкладці Properties розташовано список властивостей поточного компонента або самої форми. У вкладці Events наведено перелік подій, доступних для цього компонента.

Панель інструментів містить велику кількість візуальних та невізуальних компонентів, що дозволяє ефективно створювати програми різної складності з розвиненим інтерфейсом.

Створену порожню форму можна відразу запустити на виконання натиснувши кнопку **Run** або комбінацію клавіш Shift+Ctrl+F9.

Налаштуймо деякі властивості форми та створимо класичну програму «Hello World».

Для встановлення заголовку форми «Привіт усім» в Інспекторі об'єктів знайдіть властивість Caption і введіть відповідний текст у поле значення праворуч.

Для запуску форми у центрі екрана використайте налаштування в Інспекторі об'єктів для властивості Position та зі списку праворуч виберіть значення poScreenCenter.

Перейдіть до Панелі інструментів (Tool Palette). Розкрийте список Standard, клацніть на рядку TLabel, а потім на будь-якому місці на формі. На формі буде розміщено компонент **Мітка** (Label). Встановіть його властивість Caption у значення «Привіт, світ!». Перемістіть компонент мишею так, щоб він виглядав коректно, тобто розмістити його по центру вікна.

Створену програму можна запустити. Функціональність застосунку та якість його інтерфейсу повністю залежать від кваліфікації програміста.

5.1.2 Початок роботи над новим застосунком FMX

Щоб розпочати роботу над універсальним (`Multi-Device`) застосунком, потрібно в меню `File` вибрати команду `New → Multi-Device Application – Delphi`, а потім у діалоговому вікні (рис. 5.2) обрати шаблон програми. Шаблон задає структуру вікон програми та механізм навігації між ними. Найпростішими шаблонами є `Blank Application` та `Header/Footer`. Вони передбачають створення одновіконного застосунку. Звичайно, у процесі розробки програміст за потреби зможе додати додаткові вікна.

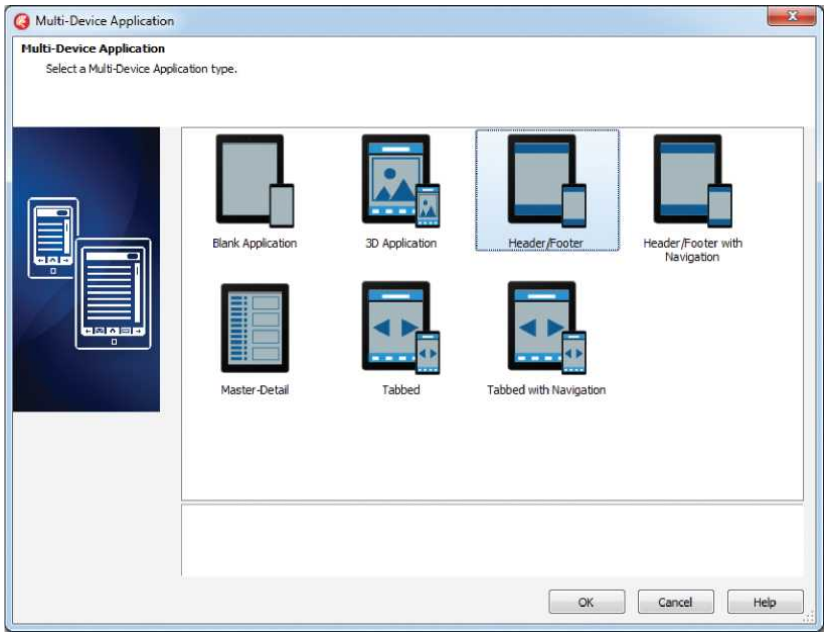

Рисунок 5.2 – У вікні потрібно вибрати шаблон (тип) програми

Оскільки проєкт є простим, всі елементи керування можна розмістити в одному вікні, тому у вікні `Multi-Device Application` доцільно обрати `Header/Footer`.

Після вибору типу програми та натискання кнопки `OK` стає доступним стандартне вікно вибору теки, у якому треба вказати каталог для проєктів

Delphi, натиснути кнопку `Створити теку` та задати ім'я каталогу проєкту, що розробляється (рис. 5.3).

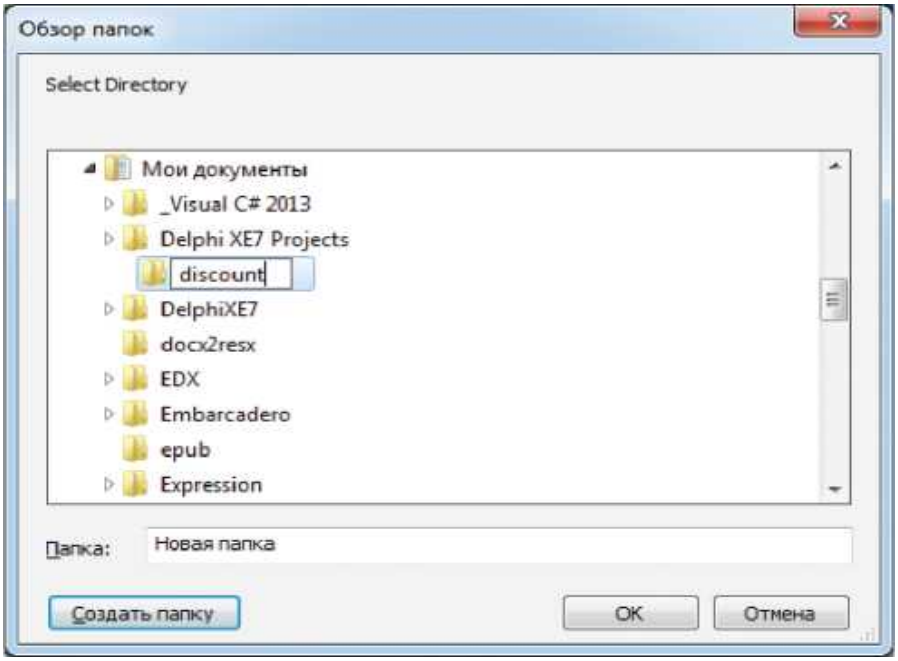

Рисунок 5.3 – Створення теці для програми

Вигляд вікна Delphi на початку роботи над новим універсальним застосунком наведено на рис. 5.4.

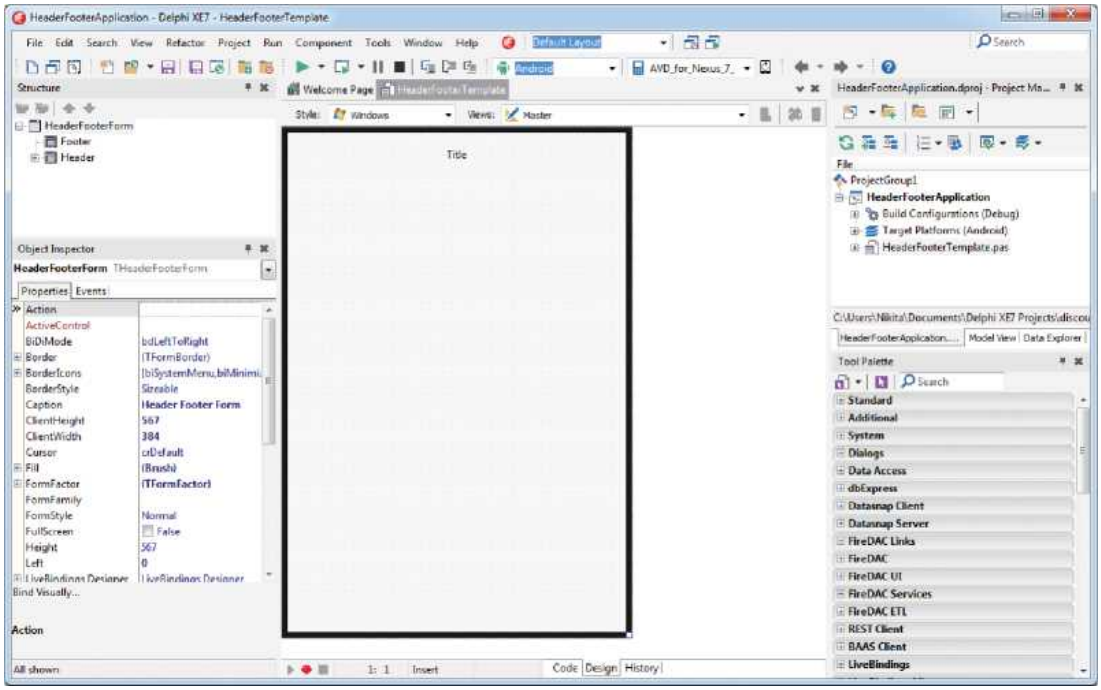

Рисунок 5.4 – Вікно Delphi на початку роботи над новим проєктом

Загалом воно мало чим відрізняється від вікна Delphi під час роботи над VCL FormsApplication. Центральну частину вікна займає конструктор форми, за яким знаходиться вікно редактора коду. Вікна **Object Inspector, Tool Palette, Project Manager** та **Structure** також на звичних місцях. Водночас слід звернути увагу на панель інструментів **Platform Device Selection** (рис. 5.5), на якій відображається поточна платформа та пристрій, для якого в цей момент йде розробка.

Рисунок 5.5 - Панель Platform Device Selection

У списку платформ перераховані ОС (рис. 5.6), на які може бути встановлений застосунок, а в списку пристроїв – реальні та віртуальні пристрої, доступні для обраної платформи.

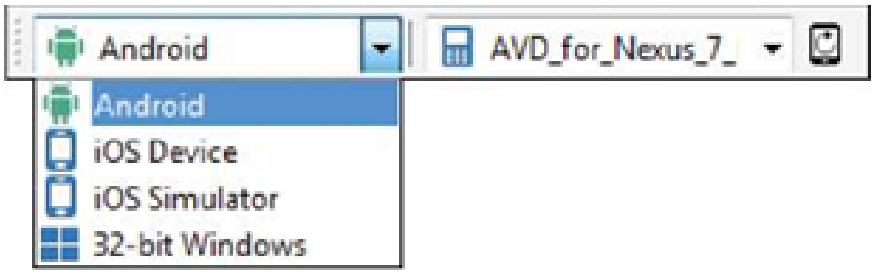

Рисунок 5.6 – Список доступних платформ

Список доступних платформ визначається конфігурацією середовища розробки, заданою під час інсталяції Delphi. Список пристроїв формується з урахуванням підключених до комп'ютера фізичних та віртуальних пристроїв (перед ім'ям віртуального Android-пристрою вказується префікс Android Virtual Device). Наведений на рис. 5.7 перелік показує, що до комп'ютера, крім кількох віртуальних Android-пристроїв, під'єднано реальний смартфон. Для

відображення фізичного пристрою у списку необхідно заздалегідь встановити відповідний USB-драйвер.

Форма та конструктор форм

Робота над універсальним застосунком починається зі створення стартової форми. Вона проєктується у спеціалізованому вікні (рис. 5.7), де розміщуються компоненти інтерфейсу з подальшим їх налаштуванням.

Рисунок 5.7 - Конструктор форм

У верхній частині вікна розташовано два спадних списка. У полі `Style` відображається назва платформи, яка визначає поточний стиль відображення форми. Вибравши потрібну платформу, можна переглянути, як виглядатиме інтерфейс на відповідному пристрої. У полі `Views` відображається назва виду для поточної платформи.

За замовчуванням для кожної платформи середовище розробки створює один універсальний (`Master`) вид. Це означає, що на всіх пристроях компоненти матимуть однакові розміри та позиціювання. Водночас для

кожного пристрою можна створити спеціальний вид (`View`), що враховує його особливості (передусім розмір екрана).

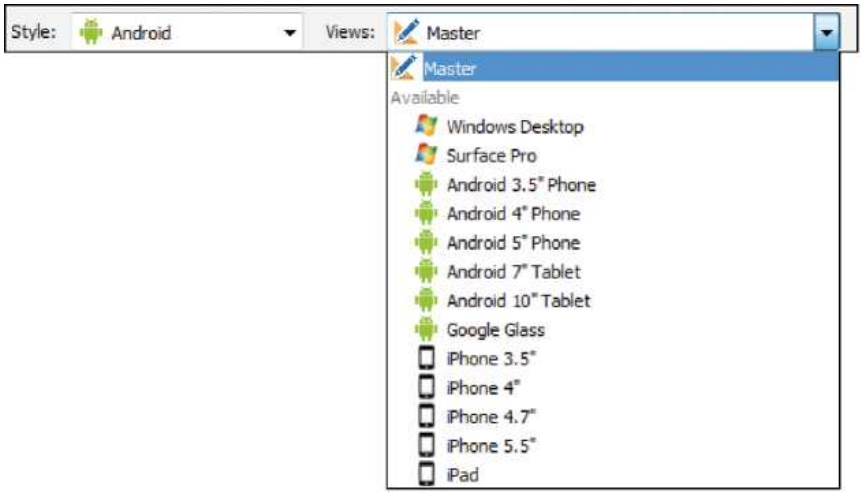

Рисунок 5.8 – Список пристроїв

У списку `Views` (рис. 5.8) перераховано пристрої, для яких програміст може налаштувати індивідуальний вигляд. Таким чином, конструктор дозволяє створити форму з єдиним набором компонентів для всієї платформи, але з власним позиціювання та масштабуванням для кожного пристрою.

Налаштування форми, як і інших компонентів, здійснюється шляхом зміни значень властивостей. При розробці Multi-Device програми в `Object Inspector` відображаються всі властивості, навіть ті, що не впливають на вигляд у поточній (цільовій) платформі. Зверніть увагу: текст `Title` у верхній частині форми – це не заголовок вікна, а текст компонента `HeaderLabel` (`TLabel`), розташованого на панелі `Header` (`TToolBar`). У нижній частині форми знаходиться ще один компонент `TToolBar`. Усі ці елементи входять до шаблону `Header/Footer`, обраного на початку роботи. Щоб змінити текст заголовка, необхідно змінити властивість `Text` компонента `HeaderLabel`. Обрати його можна клацанням миші або через дерево компонентів у вікні `Structure` (рис. 5.9).

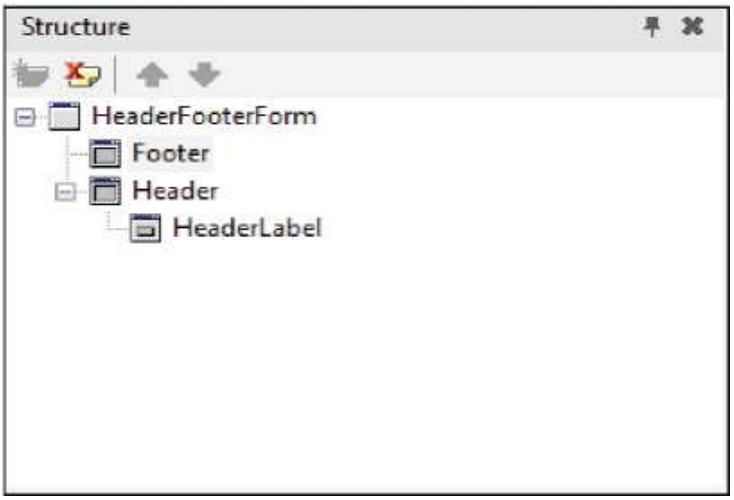

Рисунок 5.9 – Вікно Structure

5.1.3 FMX Компоненти

Компоненти для розробки розташовані на панелі компонентів. Їх вкладки та піктограми виглядають аналогічно до тих, що використовуються у VCL-застосунках. Однак, якщо навести покажчик миші на компонент, у вікні підказки (рис. 5.10) у префіксі імені модуля буде вказано FMX, а не VCL.

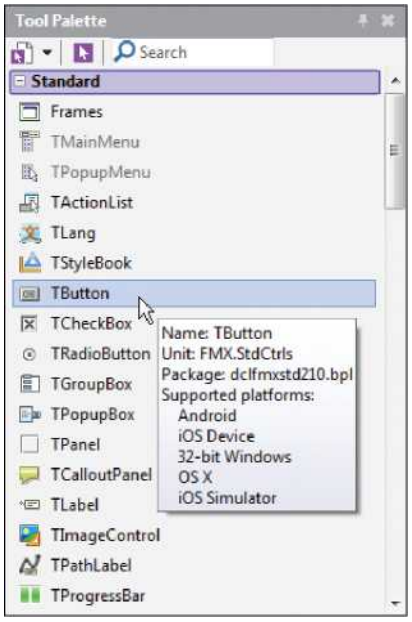

Рисунок 5.10 – FireMonkey (FMX) компоненти.

FMX – це абревіатура бібліотеки компонентів FireMonkey, яка використовується у Multi-Device застосунках. Окрім назви модуля, у вікні підказки перераховані платформи, на яких компонент підтримується. Деякі компоненти працюють не скрізь. Наприклад, WebBrowser (вкладка Internet) є доступною лише для Android та iOS.

Принципи роботи з компонентами FMX аналогічні VCL, проте мають ключові архітектурні відмінності. У FMX є властивості, яких немає у VCL, що вкрай корисні при розробці застосунків, орієнтованих на різні пристрої (зокрема, з екранами різної роздільної здатності) в межах однієї платформи. Це властивості Align, Anchors, Scale:

- Align (вирівнювання) керує розміром та позицією компонента у полі батьківського контейнера. Наприклад, якщо помістити на форму панель (TToolBar) і встановити її Align у значення Bottom, панель "прилипне" до нижньої межі форми та займе всю ширину.

- Scale дозволяє масштабувати компонент без зміни інших характеристик. Щоб збільшити текст у Label, достатньо змінити Scale.X та Scale.Y, а не розмір шрифту. Це є ефективним для адаптації під екрани з різною густиною пікселів: програма може визначити розмір екрана при запуску та автоматично налаштувати масштаб компонентів.

Окрім візуальних елементів, зручно використовувати компоненти-контейнери, які не відображаються безпосередньо, але керують розміщенням дочірніх елементів. Вони розташовані на вкладці Layouts. Найчастіше застосовують компонент-таблицю GridPanelLayout, в комірки якої можна помістити інші компоненти, та компонент Layout, що дозволяє вкласти один компонент в інший. Компонент Panel (вкладка Standard) об'єднує групу компонентів та дозволяє керувати ними як єдиним цілим.

5.1.4 Проєктування інтерфейсу

Під час створення форми універсальної програми характеристики екрана кінцевого пристрою є заздалегідь невідомими. Тому підхід до розміщення компонентів, що полягає в їх прив'язці до фіксованих координат, є неприйнятним. Наприклад, якщо програміст орієнтується на пристрій з екраном 5 дюймів, то на пристрої з екраном 4 дюйми компоненти, розташовані біля правої та нижньої меж, можуть вийти за межі видимості Також слід враховувати зміну орієнтації пристрою під час виконання, що еквівалентно зміні параметрів екрана. Щоб програма була справді універсальною, при розробці інтерфейсу слід використовувати такий підхід як **адаптивна розмітка**, де компоненти прив'язуються не до абсолютних координат (`Position.X`, `Position.Y`), а до меж контейнера або до інших компонентів. Прив'язка виконується через властивість `Align`, яка задає метод вирівнювання. Контейнер – це компонент (форма або, наприклад, `Panel`), у полі якого розміщено інший елемент. Контейнером верхнього рівня є форма. Основні значення `Align` наведено в таблиці 5.1.

Таблиця 5.1 – Значення властивості Align

Значення	**Положення компонента у контейнері**
`Top`	Притискається до верхньої межі. Ширина розтягується до ширини контейнера, висота зберігається.
`Bottom`	Притискається до нижньої межі. Ширина розтягується до ширини контейнера, висота зберігається.
`Left`	Притискається до лівої межі. Висота розтягується до висоти контейнера, ширина зберігається.
`Right`	Притискається до правої межі. Висота розтягується до висоти контейнера, ширина зберігається.
`Client`	Займає всю вільну область контейнера.
`Center`	Розташовується по центру по горизонталі та вертикалі.
`VertCenter`	Розташовується по центру вертикально. Ширина встановлюється рівною ширині вільної області.
`HorCenter`	Розташовується по центру горизонтально. Ширина не змінюється, висота встановлюється рівною висоті вільної області.

При використанні цієї технології критично важливим є порядок додавання компонентів на форму та послідовність налаштування їхніх властивостей Align.

Для позиціювання компонента всередині контейнера (зміщення щодо сусідніх елементів) слід використовувати властивість Margins, яка задає відступи меж компонента від меж контейнера.

При проєктуванні форми зручно використовувати компоненти Panel, FlowLayout, GridLayout, GridPanelLayout.

FlowLayout дозволяє розмістити кілька компонентів послідовно. Зміна ширини попереднього елемента автоматично зсуває наступні. Це ідеально для зв'язку пояснювального тексту (Label) та поля введення (Edit): при зміні тексту підказки поле введення автоматично зміститься, зберігаючи заданий відступ.

GridLayout є сіткою з комірок, корисною для розміщення однотипних компонентів (кнопки клавіатури, ескізи зображень) у вигляді матриці.

Прив'язку до меж форми також можна виконати через властивість Anchors. Наприклад, щоб кнопка залишалась у лівому нижньому куті форми незалежно від орієнтації пристрою, властивостям Anchors.akLeft та Anchors.akBottom треба присвоїти True, а Anchors.akTop та Anchors.akRight – False.

Концепція універсальності передбачає створення однієї форми, яка коректно відображатиметься на всіх пристроях. На практиці цього не завжди вдається досягти. Тому програміст може створити індивідуальні налаштування для кожного пристрою, коригуючи відступи (Margins) та масштаб (Scale) з урахуванням розміру екрана.

Методика створення форми універсального застосунку:

1. Створити Master-форму програми.

2. У списку View дизайнера форми обрати вигляд, що відповідає цільовому пристрою.

3. За потреби виконати «тонке» налаштування: змінити масштаб компонентів (`Scale`) або відступи (`Margins`).

4. Повторити кроки 2 та 3 для кожного розміру екрана, який має підтримувати застосунок.

5.2 Порядок виконання лабораторної роботи

1. Розробіть VCL-проєкт згідно з варіантом.
2. Створіть аналогічний проєкт, але з використанням FMX.

5.3 Варіанти індивідуальних завдань

1. У центрі форми розташуйте 4 кнопки, зберігши вихідні написи (див. рис. 5.11). Розробіть проєкт, у якому при натисканні на кнопку напис на ній змінюватиметься на **Вгору, Вниз, Вправо** або **Вліво**.

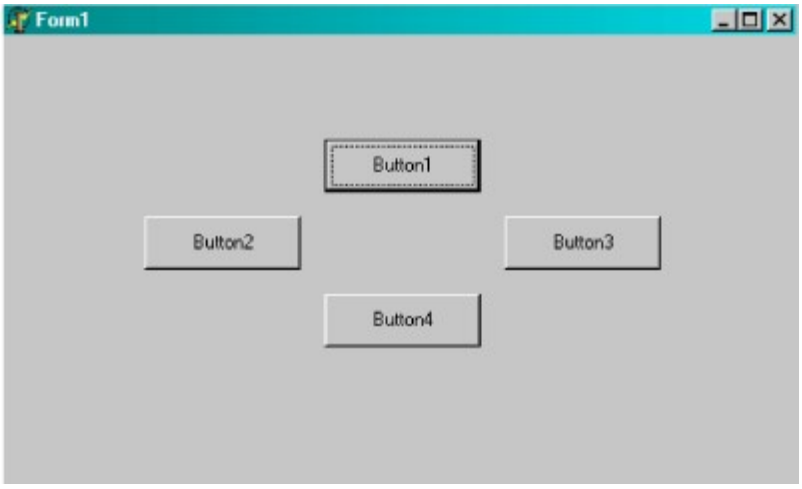

Рисунок 5.11 – Приклад виконання індивідуального завдання

2. У центрі форми розташуйте 4 кнопки, зберігши вихідні написи (див. рис. 5.11). Розробіть проєкт, у якому при натисканні на кнопку вона переміщатиметься у відповідному напрямку на задану величину.

3. У центрі форми розташуйте 4 кнопки, зберігши вихідні написи на них (див. рис. 5.11). Розробіть проєкт, у якому при натисканні на кнопку колір форми змінюватиметься.

4. У центрі форми розташуйте 4 кнопки, зберігши вихідні написи (див. рис. 5.11). Розробіть проєкт, у якому при натисканні на кнопку форма зміщуватиметься у відповідному напрямку на задану величину.

5. У центрі форми розташуйте 4 кнопки, зберігши вихідні написи (див. рис. 5.11). Розробіть проєкт, у якому при натисканні на кнопку розмір форми збільшуватиметься вліво, вправо, вниз або вгору на задану величину.

6. У центрі форми розташуйте 4 кнопки (див. рис. 5.12). Розробіть проєкт, у якому при клацанні на кнопці вона переміститься у відповідний кут форми (**ЛВ** – лівий верхній, **ПВ** – правий верхній, **ЛН** – лівий нижній, **ПН** – правий нижній).

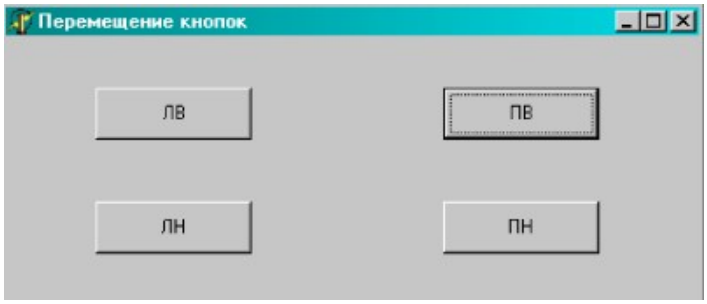

Рисунок 5.12 – Приклад виконання індивідуального завдання

7. У центрі форми розташуйте 4 кнопки (див. рис. 5.13). Розробіть проєкт, у якому при клацанні на тій чи іншій кнопці форма переміститься у відповідний кут екрана (ЛВ – лівий верхній, ПВ – правий верхній, ЛВ – лівий нижній, ПН – правий нижній).

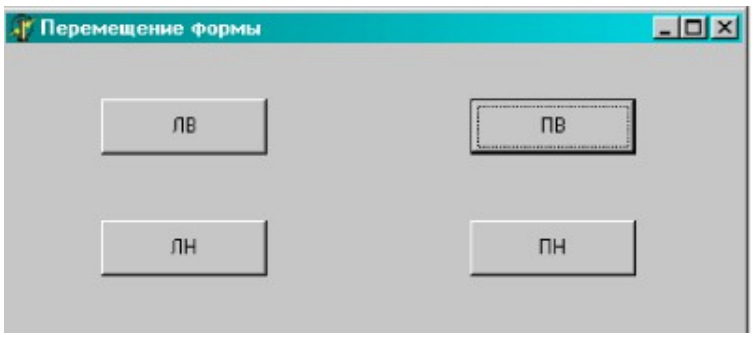

Рисунок 5.13 – Приклад виконання індивідуального завдання

8. Розробіть проєкт, у якому клацанням на тій чи іншій із двох кнопок:

- компонент **Label** стає невидимим чи видимим;
- третя кнопка стає неактивною чи активною;
- компонент **Memo** стає невидимим чи видимим;
- компонент **Edit** стає неактивним чи активним.

9. Розробіть проєкт, у якому колір компонента **Panel** змінюється шляхом встановлення інтенсивності червоного за допомогою вікна введення (див. рис. 5.14).

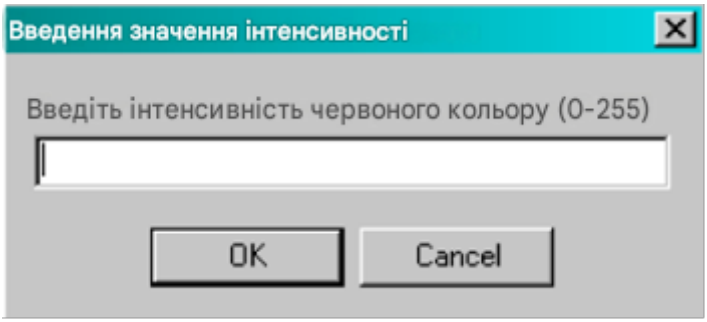

Рисунок 5.14 – Вікно введення інтенсивності червоного кольору

10. Розробіть проєкт, у якому при натисканні на кнопках відбувається збільшення або зменшення розміру шрифту напису (див. *рис.* 5.5). Вихідний розмір прийняти рівним 12 пунктів, крок зміни - 8 пунктів. При досягненні розміру шрифту 52 пунктів кнопка з написом **Збільшити** має бути відключена (див. рис. *5*.15), а при зменшенні розміру шрифту до 8 пунктів повинна бути відключена друга кнопка, при цьому активізована перша (див. рис. *5*.16).

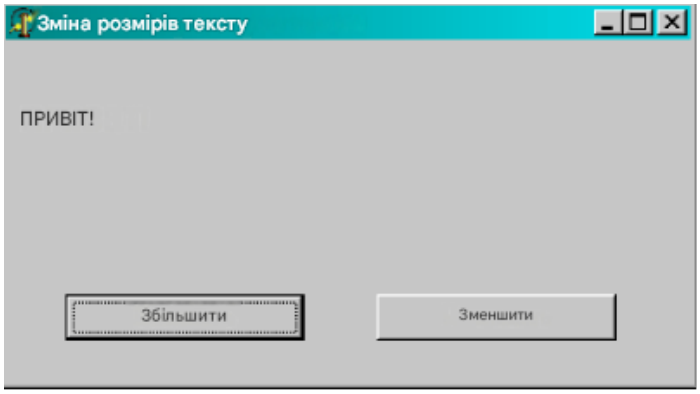

Рисунок 5.15 – Приклад виконання індивідуального завдання

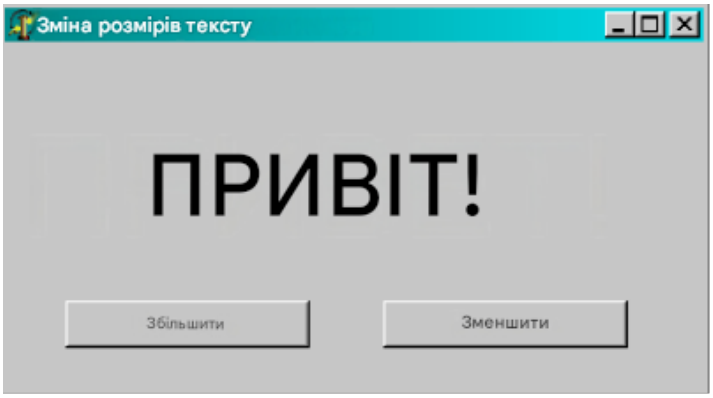

Рисунок 5.16 – Приклад виконання індивідуального завдання

5.4 Контрольні питання

1 Які властивості компонентів ви змінювали у VCL-компонентах під час створення проєкту?

2 Які властивості компонентів ви змінювали у FMX-компонентах під час створення проєкту?

3 Проведіть відповідність властивостей використаних VCL та FMX-компонентів.

4 Навіщо потрібні події візуальних компонентів? Які події Ви використали у проєкті?

6 ЛАБОРАТОРНА РОБОТА № 6 – СТВОРЕННЯ SDI-ЗАСТОСУНКУ

Мета роботи: навчитися створювати програми з SDI-інтерфейсом.

6.1 Теоретичні відомості

Термін SDI (*Single Document Interface*) дослівно означає однодокументний інтерфейс і описує програми, здатні завантажувати та використовувати одночасно лише один документ. Наприклад програма **Notepad** (наведена на рис. 6.1) є представником такого класу застосунків.

Слід уточнити термін «документ». Сучасні програми стають дедалі об'єктно-орієнтованими, тобто вони працюють із центральним об'єктом, до якого можуть бути впроваджені зовнішні об'єкти. Зазвичай ці зовнішні об'єкти обробляються іншими спеціалізованими застосунками. Наприклад програма **Wordpad** (див. рис. 6.2), що дозволяє впроваджувати будь-які OLE-об'єкти у файли. Попри це, він залишається SDI-застосунком, оскільки може працювати лише з одним об'єктом (або документом у широкому значенні цього слова).

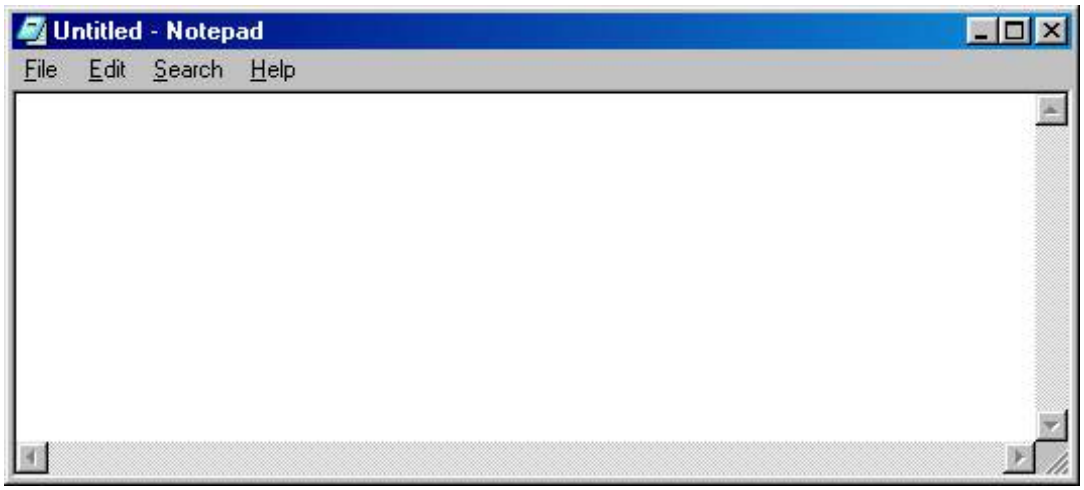

Рисунок 6.1 – Програма Notepad як приклад SDI-програми

Здатність одночасно працювати лише з одним об'єктом не заважає застосунку використовувати додаткові форми, наприклад діалогові вікна, панелі інструментів тощо (на рис. 6.2 показано панелі інструментів у вікні Wordpad). Для реалізації цих можливостей у Delphi достатньо додати форму до проєкту та встановити її властивість FormStyle у значення fsSizeToolWin або fsToolWindow.

Ще одним прикладом може бути саме середовище Delphi: величезна кількість панелей інструментів, меню, бібліотек компонентів та взаємодіючих форм. Загалом воно залишається SDI-застосунком, оскільки може завантажити та редагувати одночасно лише один об'єкт (файл коду або форми).

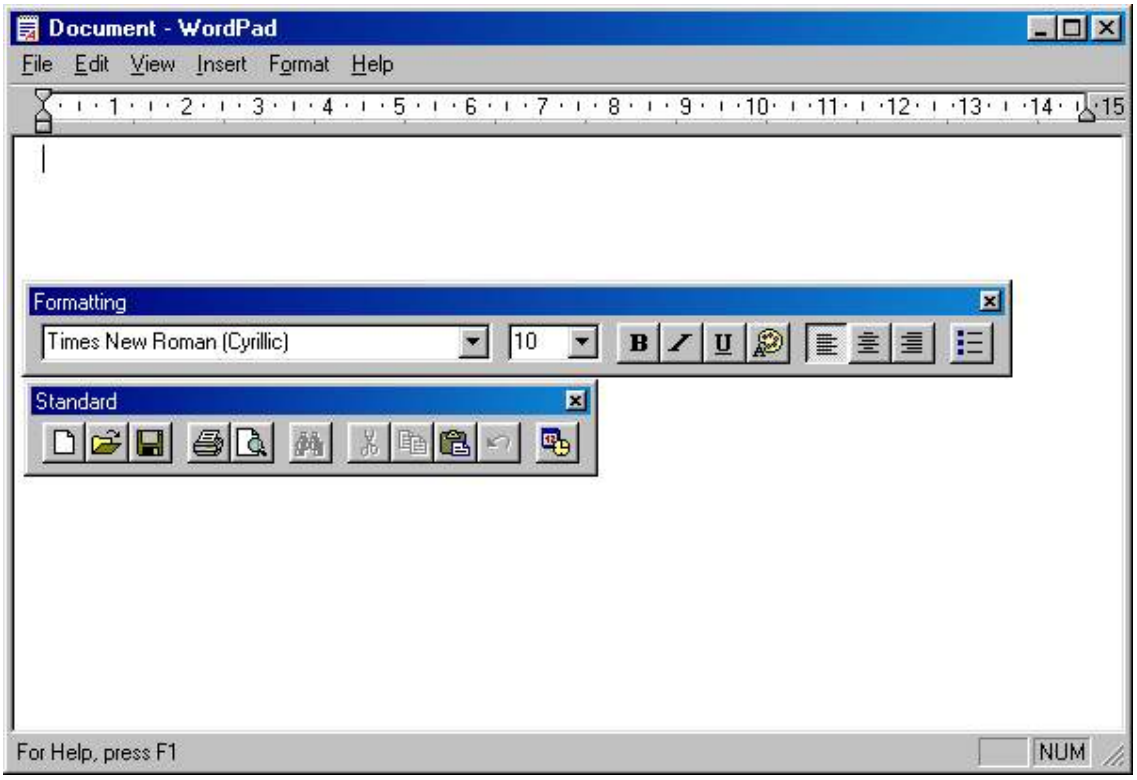

Рисунок 6.2 – Програма Wordpad – SDI- застосунок з багатьма формами

6.2 Порядок виконання лабораторної роботи

6.2.1 Приклад поетапного створення базового SDI-застосунку

Для демонстрації принципів SDI створимо просту програму відтворення зображення.

Побудова інтерфейсу:

1. Оберіть `File → New → Application`. З'явиться пустий проєкт з головною формою.
2. Встановіть властивості форми згідно з таблицею 6.1.

Таблиця 6.1 – Властивості головної форми

Властивість	**Значення**
`Caption`	`Image Viewer`
`Name`	`frmMain`
`ShowHint`	`True`

3. Помістіть на форму компонент `TPanel` та налаштуйте його властивості: `Align = alTop`, `Caption` залиште порожнім (-).

4. Додайте три компоненти `TSpeedButton` всередину `TPanel` та надайте їм імена: `spbtnLoad`, `spbtnStretch`, `spbtnCenter`. Встановіть властивості згідно з таблицею 6.2.

Таблиця 6.2 – Властивості кнопок TSpeedButton

Компонент / Властивість	**Значення**
spbtnLoad.Hint	Load
spbtnLoad.Left	8
spbtnLoad.Top	8
spbtnStretch.AllowAllUp	True
spbtnStretch.GroupIndex	1
spbtnStretch.Hint	Stretch

Компонент / Властивість	Значення
spbtnStretch.Left	48
spbtnCenter.GroupIndex	2
spbtnCenter.Hint	Center
spbtnCenter.Left	80
spbtnCenter.Top	8

5. Помістіть другий компонент `TPanel` на форму. Встановіть: `Align = alClient`, `Caption = ""`.

6. У новостворену панель помістіть компонент `TImage`. Встановіть: `Align = alClient`, `Name = imgMain`.

7. Додайте на форму компонент `TOpenDialog` з властивостями: `Filter = Bitmaps (*.bmp)|*.bmp`, `Name = opndlgLoad`, `Options = [ofPathMustExist, ofFileMustExist]`.

Delphi містить набір значків для `TSpeedButton` у каталозі `IMAGES\BUTTONS`. Збережіть проєкт через `File → Save Project As`: модуль як `Main.pas`, проєкт як `EgSDIApp.dproj`.

Написання програмного коду:

Завантаження зображення: Двічі клацніть `spbtnLoad`. В обробнику `OnClick` введіть:

```
if opndlgLoad.Execute then
 imgMain.Picture.LoadFromFile(opndlgLoad.FileName);
```

Пояснення: Метод `Execute` викликає стандартне діалогове вікно. При виборі файлу та натисканні `OK` повертає `True` і записує шлях у `FileName`. При скасуванні або натисканні `<Esc>` повертає `False`. Клас `TPicture` (властивість `Picture`) забезпечує роботу з растровими зображеннями, піктограмами та метафайлами. Метод `LoadFromFile` завантажує файл за вказаним шляхом.

Розтягування зображення: Двічі клацніть `spbtnStretch`. В обробнику `OnClick` введіть:

```
imgMain.Stretch := spbtnStretch.Down;
```

Пояснення: Властивість `Down` компонента `TSpeedButton` дорівнює `True`, коли кнопка натиснута. Властивість `Stretch` компонента `TImage` дозволяє масштабувати зображення під розмір контейнера.

Центрування зображення: Аналогічно для кнопки `spbtnCenter` введіть:

```
imgMain.Center := spbtnCenter.Down;
```

Після написання коду оберіть `Run` → `Run` для компіляції та перевірки працездатності програми.

6.2.2 Виконання індивідуального завдання (VCL)

Використовуючи алгоритм із п. 6.2.1, створіть власний проєкт відповідно до отриманого варіанта з таблиці 6.3. Забезпечте функціонал завантаження, відображення та обробки даних згідно з умовами завдання.

6.2.3 Адаптація проєкту під FireMonkey (FMX)

Створіть аналогічний проєкт, але з використанням компонентів бібліотеки FMX (*FireMonkey*). Зверніть увагу на відмінності в іменуванні компонентів, обробці подій та властивостях відображення між VCL та FMX.

6.3 Варіанти індивідуальних завдань

1. Створіть програму-переглядач простих текстових файлів.

2. Створіть, використовуючи компоненти FMX, програму-переглядач простих текстових файлів.

3. Створіть програму-переглядач зображень.

4. Створіть, використовуючи компоненти FMX, програму-переглядач зображень.

5. Створіть програму, яка завантажує таблицю чисел із текстового файлу в `StringGrid`.

6. Створіть, використовуючи компоненти FMX, програму, яка завантажує таблицю чисел з текстового файлу в `StringGrid`.

7. Створіть програму, яка завантажує пари чисел із текстового файлу та будує за ними графік.

8. Створіть, використовуючи компоненти FMX, програму, яка завантажує пари чисел з текстового файлу і будує за ними графік.

6.4 Контрольні питання

1. Що означає абревіатура SDI та в чому полягає головна особливість програм такого типу?

2. Наведіть два приклади SDI-застосунків, згадані в лабораторній роботі.

3. Чи дозволяє SDI-інтерфейс використовувати додаткові форми (діалогові вікна, панелі інструментів)? Якщо так, яку властивість форми потрібно для цього налаштувати?

4. Який компонент у `Delphi` призначений для відображення зображень і яка його властивість відповідає за розтягування картинки під розмір компонента?

5. Для чого використовується компонент `TOpenDialog` і який метод викликає стандартне вікно вибору файлу?

6. Що повертає метод `Execute` компонента `TOpenDialog` у разі успішного вибору файлу, а що – у разі скасування операції?

7. Яку властивість кнопок `TSpeedButton` потрібно перевіряти в коді, щоб дізнатися, чи кнопка натиснута, і як це пов'язано з функціями `Stretch` та `Center`?

8. Як властивість `Align` впливає на розташування компонентів? Чим відрізняються значення `alTop` та `alClient`?

9. Який фрагмент коду використовується для завантаження обраного файлу в `TImage` і чому перед цим обов'язково перевіряється результат `Execute`?

7 ЛАБОРАТОРНА РОБОТА № 7 – СТВОРЕННЯ MDI-ЗАСТОСУНКУ

Мета роботи: навчитися створювати програми з багатодокументним інтерфейсом.

7.1 Теоретичні відомості

Термін MDI (*Multiple Document Interface*) дослівно означає багатодокументний інтерфейс і описує застосунки, здатні завантажувати та використовувати одночасно кілька документів чи об'єктів. Прикладом такої програми може бути класичний диспетчер файлів (File Manager) або сучасні IDE.

Зазвичай MDI-застосунки складаються мінімум із двох форм: головної та дочірньої. Властивість головної форми FormStyle встановлюється рівною fsMDIForm. Для дочірньої форми встановлюється стиль fsMDIChild.

Головна форма служить контейнером, що містить дочірні форми. Вони розміщуються в клієнтській області батьківського вікна, можуть переміщуватися, змінювати розміри, мінімізуватися або максимізуватися. В одному застосунку можуть бути дочірні форми різних типів (наприклад, одна для обробки зображень, інша для роботи з текстом).

Створення форм у MDI-проєкті. У MDI-застосунку зазвичай потрібно виводити кілька екземплярів класу форми. Оскільки кожна форма є об'єктом, вона повинна бути створена перед використанням і звільнена, коли її більше не потребують. Delphi може робити це автоматично, або ж ви можете керувати цим процесом вручну.

Автоматичне створення форм. За замовчуванням під час запуску програми Delphi автоматично створює по одному примірнику кожного класу форм у проєкті та звільняє їх при завершенні. Цей процес обробляється у трьох місцях:

- розділ `interface` файлу модуля форми (оголошення класу):
```
type TForm1 = class(TForm) ... end;
```

- опис глобальної змінної класу:
```
var Form1: TForm1;
```

- вихідний текст проєкту (`Project → View Source`):
```
Application.CreateForm(TForm1, Form1);
```

Видалення форм обробляється через механізм власників об'єктів: коли власник знищується, автоматично знищуються всі об'єкти, якими він володіє. Створена описаним чином форма належить об'єкту `Application` і видаляється під час закриття програми.

Динамічне створення форм. Хоча автоматичне створення корисне для SDI, у MDI-застосунках воно зазвичай неприйнятне. Для створення нового екземпляра під час виконання використовуйте конструктор `Create`:

```
Form1 := TForm1.Create(Application);
Form1.Caption := 'New Form';
```

Параметр конструктора – нащадок TComponent, який стане власником форми. Зазвичай це Application (для автоматичного закриття). Можна передати Nil, створивши форму без власника, але тоді її доведеться знищувати самостійно, інакше виникне витік пам'яті.

Альтернативний варіант (без створення глобальної змінної):

```
with TForm1.Create(Application) do
Caption := 'New Form';
```

Примітка: При розробці MDI-програми метод `Show` не потрібен, оскільки `Delphi` автоматично відображає всі новостворені дочірні форми. У SDI-програмах `Show` викликати обов'язково.

Щоб відмовитися від автоматичного створення форм, відкрийте `Project` → `Options` → `Forms` та перенесіть дочірні форми зі списку `Auto-create forms` до `Available forms` (рис. 7.1).

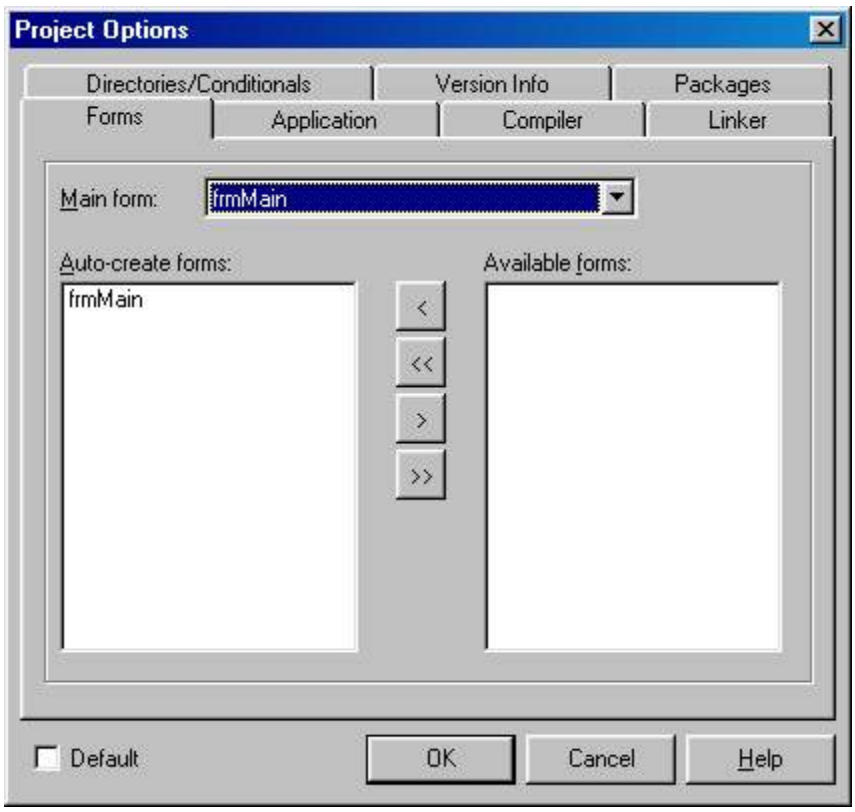

Рисунок 7.1 – Діалогове вікно Project Options

Основні властивості та методи MDI-форм

ActiveMDIChild – повертає дочірній об'єкт TForm, що має активний фокус. Корисний для панелей інструментів/меню батьківської форми.

MDIChildren та MDIChildCount – масив та кількість відкритих дочірніх форм. Використовуються для масових операцій.

TileMode – визначає спосіб розміщення дочірніх вікон під час виклику методу Tile (tbHorizontal або tbVertical).

WindowMenu – посилання на пункт меню верхнього рівня, в якому Delphi автоматично формує список активних дочірніх вікон.

Специфічні методи: ArrangeIcons, Cascade, Tile, Next/Previous.

7.2 Порядок виконання лабораторної роботи

7.2.1 Створення головної форми (VCL)

1. Оберіть `File` → `New` → `Application`.

2. Встановіть властивості форми згідно з таблицею 7.1.

Таблиця 7.1 – Властивості головної форми

Властивість	**Значення**
`Caption`	`Image Viewer`
`FormStyle`	`fsMDIForm`
`Name`	`frmMDIParent`
`ShowHint`	`True`

3. Помістіть на форму компонент `TPanel`. Встановіть: `Align = alTop`, `Caption = ""`.

4. Додайте три компоненти `TSpeedButton` у панель та назвіть їх: `spbtnLoad`, `spbtnStretch`, `spbtnCenter`. Властивості наведено в таблиці 7.2.

5. Додайте компонент `TOpenDialog` з властивостями: `Filter = Bitmaps (*.bmp)|*.bmp`, `Name = opndlgLoad`, `Options = [ofPathMustExist, ofFileMustExist]`.

7.2.2 Створення дочірньої форми (VCL)

1. Оберіть File → New → Form.

2. Встановіть властивості згідно з таблицею 7.3.

Таблиця 7.3 – Властивості дочірньої форми

Властивість	**Значення**
FormStyle	fsMDIChild
Name	frmMDIChild
Position	poDefaultPosOnly

3. Розмістіть компонент TImage на дочірній формі. Встановіть: Align = alClient, Name = imgMain.

7.2.3 Налаштування проєкту та реалізація коду

1. Видаліть дочірню форму зі списку автоматичного створення: Project → Options → Forms, перенесіть frmMDIChild до Available forms.

2. Збережіть проєкт: File → Save Project As. Модуль головної форми – MDIParent, проєкт – EgMDIApp.

Написання програмного коду:

Завантаження зображення (головна форма). В обробнику OnClick кнопки spbtnLoad введіть:

```
procedure TfrmMDIParent.spbtnLoadClick(Sender: TObject);
begin
  if opndlgLoad.Execute then begin
   var ChildForm := TfrmMDIChild.Create(Application);
```

```
      try
        ChildForm.Caption := opndlgLoad.FileName;
        ChildForm.imgMain.Picture.LoadFromFile(opndlgLoad.FileName);
        ChildForm.ClientWidth := ChildForm.imgMain.Picture.Width;
        ChildForm.ClientHeight := ChildForm.imgMain.Picture.Height;
      except
        ChildForm.Free; // Звільняємо пам'ять у разі помилки
        raise;
      end;
    end;
  end;
```

У модулі `MDIParent` після `implementation` додайте: `uses MDIChild;`

Закриття дочірньої форми: Дочірні форми за замовчуванням згортаються в піктограму. Щоб закрити їх, в обробнику `OnClose` форми `frmMDIChild` додайте: `Action := caFree;`

Розтягування та центрування: Оскільки дії застосовуються до активного вікна, код кнопок змінюється:

```
if not (ActiveMDIChild = nil) then
 if ActiveMDIChild is TfrmMDIChild then
  TfrmMDIChild(ActiveMDIChild).imgMain.Stretch :=spbtnStretch.Down;
```

Аналогічно для `spbtnCenter`.

Синхронізація стану кнопок: В обробнику `OnActivate` форми `frmMDIChild` додайте:

```
frmMDIParent.spbtnStretch.Down := imgMain.Stretch;
frmMDIParent.spbtnCenter.Down := imgMain.Center;
```

У модулі `MDIChild` після `implementation` додайте: `uses MDIParent;`

7.2.4 Імітація MDI-інтерфейсу в FireMonkey (FMX)

Бібліотека FMX не підтримує нативний MDI (властивість `FormStyle` ігнорується). Для досягнення аналогічного ефекту використовують контейнерні компоненти та динамічне створення фреймів. Нижче наведено покрокову реалізацію.

Підготовка батьківського контейнера

Створіть головну форму FMX. Додайте компонент `TLayout` (або `TScrollBox`) та встановіть: `Name = MDIContainer`, `Align = Client`. Цей компонент гратиме роль клієнтської області, в якій розміщуватимуться "вікна".

Створення та динамічне додавання дочірніх вікон

У FMX замість форм зручно використовувати `TFrame`. Створіть новий фрейм (`File → New → Other → Multi-Device Delphi Files → Frame`), назвіть `TfrmChild`. Додайте на нього `TPanel` (як заголовок) та робочу область (наприклад, `TImage`). У батьківській формі реалізуйте процедуру створення:

```
procedure TMainForm.btnNewChildClick(Sender: TObject);
var
 Child: TfrmChild;
begin
 Child := TfrmChild.Create(nil);
 Child.Parent := MDIContainer;
 Child.Position.Point := PointF(30, 30); // початкове зміщення
 Child.Size.Point := PointF(400, 300);
 Child.BringToFront; // показуємо поверх інших
end;
```

Параметр `nil` у конструкторі означає відсутність автоматичного власника. Звільнення контролюватиметься самостійно.

Реалізація переміщення та імітації фокусу

FMX не керує фокусом вікон автоматично. Перетягування реалізується через обробку подій миші на заголовку, а фокус імітується зміною візуального стану та порядку відображення.

```
procedure TfrmChild.HeaderPanelMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Single);
begin
 if Button = mbLeft then begin
  FDragging := True;
  FStartPos := PointF(X, Y);
  BringToFront; // активне вікно стає поверх інших
  // Імітація фокусу: підсвічуємо рамку
  BackgroundPanel.Stroke.Color := TAlphaColors.Crimson;
  BackgroundPanel.Stroke.Width := 2;
 end;
end;
```

Алгоритми каскадного та плиткового розміщення.

Оскільки FMX не має вбудованих методів `Cascade`/`Tile`, їх реалізують вручну, перебираючи елементи контейнера.

Коректне закриття та звільнення ресурсів.

У FMX форми не закриваються автоматично, тому звільнення контролюється програмно. Додайте на фрейм кнопку закриття (`btnClose`) або обробіть подію закриття:

```
procedure TfrmChild.btnCloseClick(Sender: TObject);
begin  Free; end;
```

7.3 Варіанти індивідуальних завдань

1. Створіть MDI-програму-переглядач простих текстових файлів.
2. Створіть MDI-програму-переглядач картинок.
3. Створіть MDI-програму, яка завантажує таблицю чисел із текстового файлу в `StringGrid`.
4. Створіть MDI-програму, яка завантажує пари чисел із текстового файлу та будує за ними графік.
5. Створіть, використовуючи компоненти FMX, MDI-програму-переглядач простих текстових файлів.
6. Створіть, використовуючи компоненти FMX, MDI-програму-переглядач картинок.
7. Створіть, використовуючи компоненти FMX, MDI-програму, яка завантажує таблицю чисел з текстового файлу в `StringGrid`.
8. Створіть, використовуючи компоненти FMX, MDI-програму, яка завантажує пари чисел з текстового файлу і будує за ними графік.

7.4 Контрольні питання

1. Як зробити форму дочірньою в середовищі `Delphi`?
2. Як програмно показати динамічно створену дочірню форму?
3. Як заховати або повністю закрити/звільнити дочірню форму під час виконання?
4. Як змінити взаємне розташування дочірніх вікон (каскад, плитка)?
5. Які стандартні діалогові форми є у бібліотеці компонентів `Delphi` та як їх викликати?

8 ЛАБОРАТОРНА РОБОТА № 8 – КОМПОНЕНТИ РОБОТИ З БАЗАМИ ДАНИХ

Мета роботи: опанувати архітектуру підключення до реляційних баз даних у середовищі Delphi за допомогою сучасного стеку FireDAC, навчитися організовувати зв'язок таблиць типу «один-до-багатьох», а також реалізувати явне керування транзакціями з урахуванням бізнес-валідації даних.

8.1 Теоретичні відомості

8.1.1 Основні компоненти

Для доступу до баз даних у Delphi традиційно використовують три групи компонентів:

1. Невізуальні компоненти: TTable, TQuery, TDataSet, TField.
2. Візуальні компоненти: TDBGrid, TDBEdit, TDBImage, TDBComboBox.
3. Сполучні компоненти: TDataSource.

Перша група призначена для керування таблицями та запитами. На панелі компонентів ці об'єкти розташовані на вкладці Data Access. Друга група забезпечує візуалізацію та редагування даних користувачем. Третя група (TDataSource) слугує мостом між невізуальними наборами даних та візуальними елементами керування.

Клас TDataSet є одним із найважливіших об'єктів для роботи з даними. Він містить абстрактні методи для безпосереднього керування записами. На фундаментальному рівні Dataset – це впорядкований набір записів, кожен з яких містить N полів, та покажчик на поточний запис.

Клас TDataSource використовується як провідник між TTable або TQuery та компонентами, що візуалізують дані. Властивість Enabled дозволяє

тимчасово від'єднати візуальні компоненти від набору даних, що значно прискорює програмне сканування записів, оскільки усуває зайве перемальовування інтерфейсу. Властивість AutoEdit визначає, чи переходить DataSet автоматично в режим редагування під час введення тексту.

TDataSource генерує три ключові події, пов'язані зі станом БД: OnDataChange, OnStateChange, OnUpdateData. Подія OnDataChange відбувається при переході на новий запис або зміні стану набору даних. OnUpdateData спрацьовує перед фіксацією змін у поточному записі. Альтернативно можна використовувати події самого набору даних: OnOpen, OnClose, BeforeInsert, AfterInsert, BeforeEdit, AfterEdit, BeforePost, AfterPost, OnCancel, OnDelete, OnNewRecord.

8.1.2 Архітектура FireDAC

Сучасна розробка клієнт-серверних та локальних застосунків для роботи з даними в Delphi повністю інтегрована у бібліотеку FireDAC, яка замінила застарілий шар Borland Database Engine (BDE). На відміну від BDE, FireDAC реалізує прямий протокольний зв'язок із серверами баз даних або локальними файловими сховищами. Архітектура побудована навколо трьох логічних рівнів: підключення, набори даних та зв'язування з інтерфейсом. Ключовим компонентом першого рівня є TFDConnection, який інкапсулює драйвер СУБД, параметри мережевого з'єднання та налаштування пулу з'єднань. Другий рівень представлений компонентами-нащадками TFDDataSet, серед яких TFDQuery виступає основним інструментом для формування SQL-запитів. Третій рівень забезпечується компонентом TFDDataSource, який слугує мостом між невізуальними наборами даних та візуальними елементами керування.

8.1.3 Підготовка структури даних та налаштування з'єднання

Для локального тестування найчастіше використовують SQLite. FireDAC підтримує її одразу ж, проте за замовчуванням ця СУБД не перевіряє цілісність зовнішніх ключів під час операцій вставки, оновлення чи видалення. Тому відразу після встановлення з'єднання необхідно активувати відповідну директиву, виконавши команду `PRAGMA foreign_keys = ON;`.

Створення таблиць виконується стандартними SQL-командами. Приклад структури для зв'язку «Відділи → Співробітники»:

```
CREATE TABLE IF NOT EXISTS Departments (
  DepID INTEGER PRIMARY KEY AUTOINCREMENT,
  Name TEXT NOT NULL
);
CREATE TABLE IF NOT EXISTS Employees (
  EmpID INTEGER PRIMARY KEY AUTOINCREMENT,
  DepID INTEGER NOT NULL,
  FullName TEXT NOT NULL,
  Position TEXT NOT NULL,
  FOREIGN KEY (DepID) REFERENCES Departments(DepID) ON DELETE
CASCADE
);
```

У Delphi налаштовується компонент `TFDConnection`: `DriverName = SQLite`, `Params.Add('Database=database.db')`. Активування: `Connected := True` або `Open`. Динамічне увімкнення підтримки зовнішніх ключів: `FDConnection1.ExecSQL('PRAGMA foreign_keys = ON;');` (розміщується одразу після підключення).

8.1.4 Навігація по записах та програмний доступ до полів

Методи First, Last, Next та Prior забезпечують послідовне переміщення між записами. Стан курсора контролюється логічними властивостями BOF (початок файлу) та EOF (кінець файлу). Метод MoveBy(Distance) дозволяє здійснювати стрибки на довільну кількість позицій.

Для читання чи зміни окремих значень використовуються властивості Fields та метод FieldByName. Через об'єкт TFDField доступні типобезпечні методи доступу: AsString, AsInteger, AsFloat, AsDateTime тощо. Наприклад: QDetail.FieldByName('FullName').AsString. Кількість доступних полів повертає властивість FieldCount.

8.1.5 Механізми редагування та валідація даних перед фіксацією

Виклик методу Edit переводить поточний запис у режим модифікації. Методи Insert та Append створюють новий запис. Після зміни полів викликається Post, який виконує валідацію, перевіряє типи даних та обмеження цілісності, і лише за умови успіху фіксує зміни у кеші FireDAC. Скасування змін виконується методом Cancel, видалення – Delete.

Критично важливою є подія BeforePost. Саме в її обробнику доцільно реалізовувати бізнес-перевірки. Якщо умова порушується, генерація виключної ситуації EDatabaseError зупиняє процес запису:

```
procedure TForm1.QDetailBeforePost(DataSet: TDataSet);
var
 LPos: string;
begin
 if QDetail.Fields.FindField('Position') <> nil then
 begin
  LPos := Trim(QDetail.FieldByName('Position').AsString);
```

```
       if LPos = '' then
        raise  EDatabaseError.Create('Помилка:  поле  "Посада"  не
може бути порожнім!');
      end;
     end;
```

8.1.6 Організація зв'язку Master-Detail у FireDAC

Детальний запит посилається на джерело даних головної таблиці через властивість MasterSource. У властивості MasterFields вказується ім'я поля зв'язку. При зміні поточного запису в головній таблиці FireDAC автоматично генерує новий SQL-запит для деталі, підставляючи поточне значення ключа як параметр (WHERE MasterID = :MasterID).

Порядок ініціалізації:

1. Головний запит отримує SELECT * FROM MasterTable та активується.

2. Детальний запит отримує SELECT * FROM DetailTable WHERE MasterID = :MasterID, прив'язується до MasterSource та MasterFields, і лише потім активується. Це гарантує коректне значення параметра при першому відкритті.

8.1.7 Явне керування транзакціями

У FireDAC транзакції керуються на рівні TFDConnection. Початок сесії: StartTransaction, фіксація: Commit, скасування: Rollback.

```
     procedure TForm1.btnStartClick(Sender: TObject);
     begin
      if not FDConnection1.InTransaction then
      begin
       FDConnection1.StartTransaction;
       StatusBar1.SimpleText :=' Транзакцію розпочато. Зміни ще не
збережені.';
```

```
      end;
     end;

     procedure TForm1.btnCommitClick(Sender: TObject);
     begin
      if FDConnection1.InTransaction then
      begin
       try
        FDConnection1.Commit;
        StatusBar1.SimpleText :=' Зміни успішно збережено в БД.';
        QMaster.Refresh; QDetail.Refresh;
       except
        on E: Exception do ShowMessage('Помилка фіксації:' +
E.Message);
       end;
      end;
     end;

     procedure TForm1.btnRollbackClick(Sender: TObject);
     begin
      if FDConnection1.InTransaction then
      begin
       FDConnection1.Rollback;
       StatusBar1.SimpleText := 'Всі зміни скасовано. Дані
відновлено.';
       QMaster.Refresh; QDetail.Refresh;
      end;
     end;
```

Синхронізація інтерфейсу реалізується в події `AfterScroll`:

```
     procedure TForm1.QMasterAfterScroll(DataSet: TDataSet);
     begin
      lblMasterInfo.Caption := Format('Обрано запис ID: %d | Стан:
%s',
       [QMaster.FieldByName('DepID').AsInteger,
DataSet.StateDesc]);
      btnCommit.Enabled := FDConnection1.InTransaction;
      btnRollback.Enabled := FDConnection1.InTransaction;
      btnStart.Enabled := not FDConnection1.InTransaction;
     end;
```

Ініціалізація компонентів виконується в `FormCreate`, звільнення ресурсів – у `FormDestroy`.

8.2 Порядок виконання лабораторної роботи

1. Створіть новий VCL- застосунок у середовищі Delphi та збережіть проєкт в окремій робочій теці.

2. Додайте на форму невізуальні компоненти: один TFDConnection, два TFDQuery (QMaster, QDetail), два TFDDataSource (DSMaster, DSDetail). Розмістіть візуальні елементи: два TDBGrid та три кнопки TButton (btnStart, btnCommit, btnRollback).

3. Налаштуйте підключення до SQLite, активуйте з'єднання та виконайте `PRAGMA foreign_keys = ON;` через метод `ExecSQL`.

4. Налаштуйте головний запит (`SELECT * FROM MasterTable`), зв'яжіть `DSMaster` з `QMaster`, а сітку з `DSMaster`.

5. Налаштуйте детальний запит (`SELECT * FROM DetailTable WHERE MasterID = :MasterID`), вкажіть `ParamCheck := True`, налаштуйте зв'язок `MasterSource := DSMaster` та `MasterFields := 'MasterID'`. Активуйте та зв'яжіть з сіткою.

6. Реалізуйте обробники OnClick для транзакцій (StartTransaction, Commit, Rollback) та логіку блокування кнопок в AfterScroll.

7. Реалізуйте валідацію в BeforePost детального запиту. Протестуйте програму, зробіть скриншоти та оформіть звіт.

8.3 Варіанти індивідуальних завдань

У всіх наданих варіантах таблиці пов'язані із полем ID головної таблиці (відповідає MasterID у деталі). Всі операції додавання, зміни чи видалення мають виконуватися в одній явній транзакції. При порушенні вказаної умови або виявленні помилки даних обов'язково викликається Rollback.

1. Відділи → Співробітники

Головна таблиця: Departments (DepID, Name). Детальна таблиця: Employees (EmpID, DepID, FullName, Position).

Завдання: додати новий відділ та одразу двох співробітників в одній транзакції. Перевірити, що поле Position не порожнє. Якщо умова не виконана – виконати Rollback.

2. Замовники → Замовлення

Головна таблиця: Customers (CustID, CompanyName). Детальна таблиця: Orders (OrderID, CustID, OrderDate, Sum).

Завдання: змінити назву компанії у головній таблиці та додати нове замовлення в детальній. Якщо сума замовлення Sum < 0 – відкотити транзакцію.

3. Категорії товарів → Товари

Головна таблиця: Categories (CatID, Title). Детальна таблиця: Products (ProdID, CatID, Name, Price).

Завдання: видалити категорію та всі пов'язані з нею товари в одній транзакції. Переконатися, що після Rollback дані повністю відновлюються в обох таблицях.

4. Автори → Книги

Головна таблиця: Authors (AuthID, FullName). Детальна таблиця: Books (BookID, AuthID, Title, PublishYear).

Завдання: додати автора та три книги в транзакції. Якщо рік видання будь-якої книги менший за 1900 – скасувати операцію через Rollback.

5. Навчальні групи → Студенти

Головна таблиця: Groups (GroupID, Name, Course). Детальна таблиця: Students (StudID, GroupID, FullName, AvgScore).

Завдання: оновити назву групи та змінити середній бал усіх студентів у ній на +0.5 в одній транзакції. Перевірити коректність Commit та відображення оновлених даних.

6. Проєкти → Завдання

Головна таблиця: Projects (ProjID, Title, Status). Детальна таблиця: Tasks (TaskID, ProjID, Description, Deadline).

Завдання: змінити статус проєкту на "В роботі" та додати два завдання в транзакції. Якщо дедлайн хоча б одного завдання вже минув – виконати Rollback.

7. Склади → Товари на складі

Головна таблиця: Stores (StoreID, Location). Детальна таблиця: Inventory (ItemID, StoreID, ProductName, Qty).

Завдання: "перемістити" товар (оновити StoreID та зменшити Qty в одному записі, додати/оновити в іншому) в одній транзакції. У разі порушення цілісності або від'ємної кількості – Rollback.

8. Типи номерів → Бронювання

Головна таблиця: RoomTypes (TypeID, Name, BasePrice). Детальна таблиця: Bookings (BookID, TypeID, GuestName, TotalCost).

Завдання: створити бронювання. Якщо TotalCost < BasePrice вибраного типу номера – виконати Rollback та вивести повідомлення.

9. Клієнти → Платежі

Головна таблиця: Clients (CustID, FullName, CurrentBalance). Детальна таблиця: Payments (PayID, CustID, Amount, PayDate).

Завдання: додати запис про платіж та програмно оновити CurrentBalance клієнта. Якщо після операції баланс стає від'ємним – відкотити транзакцію.

10. Замовники → Замовлення на товари

Головна таблиця: Clients (CustID, Company). Детальна таблиця: Orders (OrdID, CustID, Product, Qty).

Завдання: змінити назву компанії-замовника та додати нове замовлення в одній транзакції. Якщо Qty <= 0 або поле порожнє – виконати Rollback.

11. Університет

Головна таблиця: Факультети (FacID, Name) Детальна таблиця: Студенти (StudID, FacID, FullName, GPA) Завдання: додати новий факультет та одразу

зарахувати двох студентів в одній транзакції. Перевірити, щоб GPA було в діапазоні 0..5. При порушенні умови – виконати Rollback.

12. Бібліотека

Головна таблиця: Автори (AuthID, FullName) Детальна таблиця: Книги (BookID, AuthID, Title, PublishYear) Завдання: оновити ім'я автора та додати нову книгу в одній транзакції. Якщо PublishYear > поточного календарного року – скасувати операцію через Rollback.

13. Транспортне підприємство

Головна таблиця: Водії (DriverID, FullName, LicenseExpiry) Детальна таблиця: Рейси (TripID, DriverID, Route, TripDate) Завдання: призначити водія на новий рейс. Якщо дата закінчення ліцензії (LicenseExpiry) менша за дату рейсу – виконати Rollback.

14. Поліклініка

Головна таблиця: Лікарі (DocID, Specialty) Детальна таблиця: Приймання (VisitID, DocID, PatientName, Diagnosis) Завдання: додати лікаря та зареєструвати перший прийом. Якщо поле Diagnosis залишається порожнім після заповнення форми – відкотити транзакцію.

15. Житловий комплекс

Головна таблиця: Будинки (BuildID, Address, MaxFloors) Детальна таблиця: Квартири (AptID, BuildID, Number, Floor) Завдання: додати будинок та 3 квартири в одній транзакції. Якщо значення Floor у будь-якій квартирі перевищує MaxFloors будинку – виконати Rollback.

16. HR-система

Головна таблиця: Підрозділи (DeptID, Name) Детальна таблиця: Працівники (EmpID, DeptID, FullName, Salary) Завдання: створити новий підрозділ та додати двох працівників. Якщо Salary хоча б одного працівника нижче встановленого мінімуму – скасувати всі зміни через Rollback.

17. Складський облік

Головна таблиця: Категорії (CatID, Title) Детальна таблиця: Товари (ProdID, CatID, Name, Qty) Завдання: видалити категорію та всі пов'язані з нею

товари в одній транзакції. Перевірити коректність фіксації змін (Commit) та повне відновлення даних в обох таблицях після Rollback.

18. Готельний бізнес

Головна таблиця: Типи номерів (TypeID, Name, BasePrice) Детальна таблиця: Бронювання (BookID, TypeID, GuestName, TotalCost) Завдання: створити бронювання. Якщо TotalCost < BasePrice вибраного типу номера – виконати Rollback та вивести повідомлення про помилку ціноутворення.

19. Банківські операції

Головна таблиця: Клієнти (CustID, FullName, CurrentBalance) Детальна таблиця: Платежі (PayID, CustID, Amount, PayDate) Завдання: додати запис про платіж та програмно оновити CurrentBalance клієнта. Якщо після операції баланс стає від'ємним – відкотити транзакцію.

20. Виробниче замовлення

Головна таблиця: Замовники (CustID, Company) Детальна таблиця: Замовлення (OrdID, CustID, Product, Qty) Завдання: змінити назву компанії-замовника та додати нове замовлення в одній транзакції. Якщо Qty <= 0 або поле порожнє – виконати Rollback.

8.4 Контрольні питання

1. Які основні компоненти FireDAC використовуються для підключення до бази даних та виконання SQL-запитів, і яку роль відіграє TFDDataSource у цій архітектурі?

2. Яким чином здійснюється навігація по записах у TFDQuery та як програмно перевірити досягнення меж набору даних?

3. Який алгоритм явного керування транзакціями у FireDAC (StartTransaction, Commit, Rollback) та у яких бізнес-сценаріях його обов'язково застосовувати замість автоматичної фіксації?

4. Як реалізувати зв'язок Master-Detail у FireDAC за допомогою властивостей MasterSource та MasterFields, і яким чином драйвер обробляє параметр :MasterID?

5. Які методи TFDDataSet використовуються для початку редагування, фіксації змін та скасування незаписаних даних, і в який момент спрацьовує подія BeforePost?

6. Чому підключення до SQLite вимагає виконання PRAGMA foreign_keys = ON; і які наслідки має ігнорування цієї команди при роботі зі зв'язаними таблицями?

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Артюхов В. Г. Технології комп'ютерного проєктування. Комп'ютерний практикум : навч. посіб. / В. Г. Артюхов [та ін.]. Київ : КПІ ім. І. Сікорського, 2022. 122 с. ISBN 978-617-7919-36-4.

2. Hodges N. Coding in Delphi / N. Hodges. [S. l.] : Nepeta Enterprises, 2014. 242 p. ISBN 978-0-9899185-0-8.

3. Cantù M. Delphi Developer's Handbook / M. Cantù. San Francisco : Sybex Inc, 2001. 1134 p. ISBN 978-0-7821-4120-6.

4. Cantù M. Object Pascal Handbook: Delphi 10.4 Sydney Edition : the complete guide to the Object Pascal programming language for Delphi 10.4 Sydney / M. Cantù, 2021. 534 p. ISBN 978-8-894317-31-3. URL: https://www.marcocantu.com (дата звернення: 19.04.2026).

5. Jensen C. Delphi in Depth: FireDAC / C. Jensen. [S. l.] : Independent Publishing Platform, 2017. 556 p. ISBN 978-1-5449-0813-9.

6. Harmon E. Delphi COM Programming / E. Harmon. [S. l.] : Macmillan Technical Publishing, 2013. 512 p. ISBN 978-1-57870-210-7.

7. Безменов М. І. Основи програмування у середовищі Delphi : навч. посіб. / М. І. Безменов. Харків : НТУ «ХПІ», 2010. 608 с. ISBN 978-966-593-791-3.

8. Усатенко Т. М. Програмування в середовищі Delphi : навч. посіб. / Т. М. Усатенко. Суми : СумДУ, 2004. 84 с.