РОЗДІЛ 3. ОБ'ЄКТНО-ОРІЄНТОВАНІ ІНТЕРФЕЙСИ КОРИСТУВАЧА. ЗАСОБИ РОЗРОБЛЕННЯ ІНТЕРФЕЙСІВ КОРИСТУВАЧА

3.1 Поняття об'єктно-орієнтованого інтерфейсу користувача

3.1.1 Особливості об'єктно-орієнтованого інтерфейсу користувача

Сучасні ГІК та об'єктно-орієнтовані ІК (ООІК) суттєво відрізняються від інтерфейсів, що розроблялись 10-20 років тому. Вони значно простіші у використанні, однак користувачам доводиться витрачати чимало зусиль для вивчення документації, яка дозволяє рухатись еволюційним ланцюжком інтерфейсів. Крім того, необхідні знання загальних принципів програмування для ефективної побудови складних, керованих подіями, дружніх по відношенню до користувача програм.

Тип інтерфейсу визначає, яким чином будуть застосовуватись обчислювальні потужності комп'ютерних апаратних засобів. За словами Бада Тріббла, віце-президента по розробці ПЗ компанії NEXT Incorporated, “єдиною річчю, на яку варто витрачати комп'ютерну потужність, є інтерфейс” [25].

ООІК є засобами виведення середовища ГІК за межі простого представлення додатків і файлів у вигляді іконок на ерані комп'ютера. ГІК є інтерфейсом “стану дії”, а в ООІК користувач чітко розпізнає і маніпулює об'єктами. ООІК здається імітацією, а не відображенням реальності. Перехід від ГІК до ООІК вніс зміни на панель меню, оскільки вікна тепер стали вікнами об'єктів, а не додатків.

Існує суттєва різниця в семантиці ГІК та ООІК. В ГІК іконки на екрані лише представляють програми і файли даних. В ООІК іконки на екрані представляють об'єкти, які мають задані відношення з іншими об'єктами. Користувачі можуть побудувати концептуальні моделі інтерфейсу, працюючи безпосередньо з об'єктами, а не з додатками.

Пряме маніпулювання (метод drag-and-drop), яке є основним принципом ООІК, дозволяє без особливих зусиль

виконувати дії загального характеру безпосередньо над об'єктами: копіювання, створення, переміщення, об'єднання і т.і.

До складу основних компонентів ООІК входять усі характеристики ГІК. Часто користувачу складно виділити відмінності між ГІК та ООІК. Спочатку ГІК та ООІК здаються однаковими, головні відмінності містяться у внутрішніх моделях інтерфейсу. Основна характеристика ООІК полягає в тому, що вони не орієнтовані на додатки, а концентруються на побудові моделей користувача, які переносяться з реального світу в комп'ютерне середовище. Мета ООІК полягає у звільненні користувача від необхідності вивчення способів використання додатків і файлів.

Більшість вікон ГІК мають панелі меню, які містять однакові команди (пункти). Проблемно-орієнтована панель меню ГІК носить назву FEVH-панелі (FEVH - File, Edit, View, Help). Найпоширенішим стилем взаємодії в ООІК є послідовність "об'єкт-дія". Об'єктно-орієнтована панель меню для вікна має назву WOSH (WOSH - Window, Object, Selected objects, Help). Розглянемо реалізацію панелі меню ООІК на прикладі вікна об'єкту календаря (рис.3.1.1).

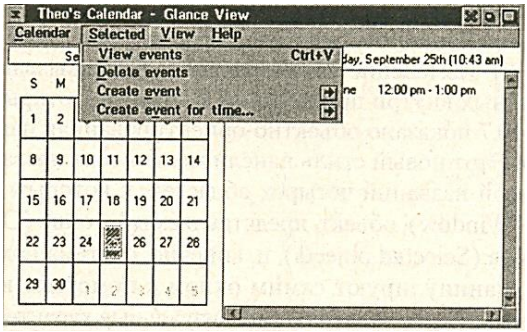

Рис.3.1.1 Пункти панелі меню для вікон об'єктів

Користувачі маніпулюють самим вікном за допомогою спадного меню System, яке містить пункти, що змінюють візуальні характеристики вікон та інші пункти, специфічні для ОС. Пункт File панелі меню та підменю змінюють вигляд для забезпечення варіантів відображення об'єкта, представленого вікном, оскільки концепції файлу більше не існує. Мітка пункта панелі меню тепер є

назвою класу або внутрішнього базового об'єкту. Наприклад, календар на рис.3.1.1 показує Calendar в якості першого пункту панелі меню. Підменю пропонує дії з об'єктом календаря в цілому. Всі пункти підменю тепер діють на сам об'єкт, а не на інформацію або об'єкти, які містяться у вікні. Другий пункт панелі меню календаря має мітку Selected і використовується для контейнера, а також об'єктів даних. Об'єкти всередині вікна обираються, і над ними виконуються відповідні дії. Меню динамічно змінюється в залежності від кожного типу обраного об'єкта. Даний пункт панелі меню є новим для ООІК. Основна його перевага в тому, що користувачам не потрібно відкривати об'єкт, який вони бачать у вікні, щоб виконувати над ним будь-яку дію. Інші пункти панелі меню та зв'язані з ними підменю для вікон об'єктів багато в чому аналогічні використовуваним для вікон додатків. Пункт View панелі меню календаря змінює тип представлення або його характеристики в межах вікна. Виклик системи допомоги виконує пункт Help панелі меню календаря.

Об'єкти є елементами, якими можна маніпулювати як цілими частинами, однак вони можуть складатись з декількох інших об'єктів, що певним чином взаємодіють між собою. Користувачі ООІК фокусують свою увагу на об'єктах, а не на додатках. Основні типи об'єктів інтерфейсу складають фундаментальні класи всіх об'єктів, які забезпечуються системою. Вони також є базою для об'єктів, розроблених на вимоги користувачів до окремого продукту, а також для користувачів при створенні ними власних об'єктів. Існує 3 типи об'єктів: об'єкти-дані, об'єкти-контейнери та об'єкти-пристрої.

Об'єкти-дані надають користувачам інформацію. Вони можуть представляти будь-який тип інформації, з якою працюють користувачі (текст, електронні таблиці, зображення, музика, відео, анімація).

Об'єкти-контейнери можуть зберігати і групувати будь-які об'єкти, в тому числі й інші контейнери, представляючи їх вміст різними способами, переміщуючи і копіюючи об'єкти з та в контейнери, а також вибудовуючи та сортуючи вміст у будь-якому порядку. Типові контейнери - папки, кошики вхідних та вихідних

повідомлень для пошти, робочий стіл і т.і. Контейнери загального призначення сприяють організації роботи користувачів у відповідності до їхніх моделей. Об'єкти загального призначення - папки і робочі області можуть вдосконалюватись з метою створення більш потужних контейнерів для зберігання специфічних продуктів. Іконки відображають об'єкти інтерфейсу і є допоміжними елементами. Візуальні контейнери, наприклад, папки, також допомагають організувати та систематизувати інформацію у пам'яті комп'ютера. Використання контекстної довідки забезпечує доступ до довідкової інформації для обраної області екранної форми та елементів управління.

Об'єкти-пристрої — пристрої, які існують в реальному світі. Будь-який предмет, з яким працює користувач, може бути відображений у вигляді об'єкта-пристрою (телефон, факс, принтер).

Представлення об'єктів дозволяють розглядати об'єкт та інформацію, яка в ньому міститься, різними способами. Перевага представлень об'єктів базується на можливості одночасно відкривати декілька представлень одного й того ж об'єкта. Представлення об'єктів повинні бути динамічними й тісно взаємозв'язаними. Коли користувачі вносять в об'єкт зміни, які впливають на інші його представлення, то новації повинні відображатись негайно чи якомога швидше. Типи представлень об'єктів подано в таблиці 3.1.1.

Таблиця 3.1.1 - Типи представлень об'єктів

Тип представлення	**Опис представлення**
Складені представлення	Відображають інформацію та об'єкти, показуючи їх порядок та взаємовідношення з іншими компонентами
Представлення змісту	Відображає компоненти або вміст об'єктів
Представлення властивостей	Дозволяють переглядати та змінювати інформацію або властивості об'єктів
Представлення системи допомоги	Надають користувачам допомогу в тому вигляді, у якому вона їм потрібна

Однією з першочергових задач ООІК є активне застосування принципів розроблення, що розвантажують пам'ять користувача. ОС надає списки елементів, а також графічні керуючі елементи, які дозволяють користувачам розпізнавати об'єкти та здійснювати свій вибір, а не згадувати та вводити інформацію.

В ООІК застосовується пряме маніпулювання, тому дуже важливо підтримувати зворотній зв'язок з користувачами за обраними операціями. При проведенні операцій прямого маніпулювання користувачі повинні бути забезпечені 4 типами зворотнього зв'язку: виділення джерела; виділення цілі; картинкою, що перетягується; вказівником миші.

Важливим етапом в підвищенні практичності та функціональності програм є можливість для користувачів взаємодіяти з кожним аспектом ООІК. Також необхідно передбачити множину способів інтерактивних дій, наприклад, використання клавіатури або миші при вирішенні задачі або переключення з одного пристрою введення інформації на інший.

3.1.2 Порівняння проблемно-орієнтованих та об'єктно-орієнтованих інтерфейсів користувача.

Інтерфейси ПЗ розвиваються від проблемно-орієнтованих до об'єктно-орієнтованих.

Відмінності між проблемно-орієнтованими та об'єктно-орієнтованими ІК наведені в таблиці 3.1.2.

Таблиця 3.1.2 - Відмінності між проблемно-орієнтованими та об'єктно-орієнтованими ІК [9]

Проблемно-орієнтовані ІК	**Об'єктно-орієнтовані ІК**
(1)	**(2)**
Додаток складається з іконки, первинного вікна та вторинних вікон	Продукт складається з набору взаємодіючих об'єктів або видів об'єктів
Іконки представляють додатки або відкриті вікна	Іконки представляють об'єкти, якими можна маніпулювати
Зміст показано за допомогою текстових полів зі списками	Папки і довідники є візуальними контейнерами

(1)	(2)
Користувачі повинні запустити додаток, перш ніж перейти до роботи з об'єктами	Користувачі відкривають представлення об'єктів на Робочому столі
Забезпечують користувачів функціями, необхідними для виконання задач	Забезпечують користувачів об'єктами, необхідними для виконання задач
Акцент робиться на основну задачу	Акцент робиться на вхідні і вихідні дані для об'єктів і задач
Взаємозв'язок задач підтримується іншими додатками	Взаємозв'язок задач підтримується використанням інших об'єктів
Жорстка структура визначається функцією	Гнучка структура визначається об'єктом
Користувач не може перервати виконання задачі	Користувач може перервати виконання задачі
Увагу користувача сфокусовано на додатку та його функціях	Увага користувача фокусується на спільних концепціях, представленнях та функціях
Користувачі повинні слідувати структурі додатку	Користувачі можуть виконувати задачі в будь-якій послідовності або вдосконалювати процес виконання
Потрібно багато додатків - по одному на задачу	Повторне використання одних і тих же об'єктів в багатьох задачах

Основні спільні проблеми графічного та об'єктно-орієнтованого інтерфейсів користувача показані на рис. 3.1.2.

Рис.3.1.2 - Спільні проблеми графічного та об'єктно-орієнтованого інтерфейсів користувача

3.1.3 Налагодження об'єктно-орієнтованого інтерфейсу користувача

ООІК повинен забезпечувати можливість налагодження практично будь-якого елемента. Одна з основних характеристик ООІК полягає в тому, що користувачі можуть налагодити кожен об'єкт цілком індивідуально на додаток до персоналізації всієї системи в цілому. ГІК звичайно забезпечує властивості та налагодження лише на рівні системи або додатку.

Операційна система Windows передбачає представлення властивостей всіх типів об'єктів. ОС OS/2 забезпечує єдиний спосіб налагодження кольорів та шрифтів за допомогою палітри шрифта, кольору та схеми кольорів. В OS/2, а також у Windows 7 і Windows Vista користувачі працюють безпосередньо з об'єктами, змінюючи їх властивості переміщенням обраного шрифта, кольору та кольорової схеми на об'єкт. Вони можуть налагоджувати індивідуальні об'єкти чи вікна або вносити зміни в усю систему.

Саме в створенні об'єктів, що слідують загальним метафорам, пов'язаним з іншими, а також із заданими властивостями та поведінкою, і полягає суть розроблення ООІК. ГІК займаються представленням та інтерактивними елементами, а ООІК забезпечують їх відповідність рівню взаємовідношень між його об'єктами.

3.2 Засоби розроблення інтерфейсів користувача у середовищі C++ Builder

3.2.1 Модель візуальних компонентів

Компоненти - це об'єкти, які можна розташовувати на екранну форму і керувати ними за допомогою властивостей, методів та подій. Такий стиль називають візуальним програмуванням.

Властивості компонентів визначають їх функції. Деякі властивості однакові для різних компонентів. Наприклад, усі візуальні компоненти мають властивості `Top` і `Left`, які визначають положення компонента відносно форми, а для форми - її положення на екрані. Всі компоненти мають властивість `Owner`, яка використовується для відстежування компонентів-нащадків. Інспектор об'єктів C++ Builder призначений для забезпечення простого і зручного інтерфейсу для зміни властивостей об'єктів C++Builder та керування подіями, на які реагує об'єкт. Він виглядає наступним чином (рис.3.2.1). Вкладка `Properties` дозволяє переглянути список властивостей об'єкта в алфавітному порядку.

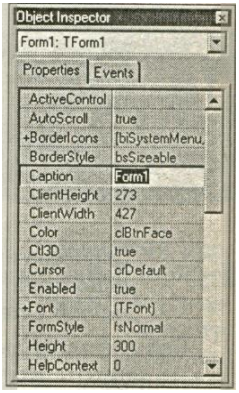

Рис. 3.2.1 Вікно інспектора об'єктів

Кожній властивості відповідає певний член класу, але самі властивості не є членами-даними. Зміна властивості призводить до неявного виконання деякого фрагменту коду. Властивості часто

пов'язані з методами доступу, які виконуються при зміні властивостей.

Властивості мають значення за замовчуванням, які відображаються у вікні інспектора об'єктів при першому використанні компонента. Значення властивостей можуть змінюватись під час розробки і під час виконання програми. В C++ Builder на екрані одразу відображається результат присвоєння властивості нового значення під час розроблення, що є безумовною перевагою цього середовища. Для зміни значення властивості під час виконання програми необхідно виконати відповідне присвоєння (метод доступу для даної властивості буде викликатись автоматично), наприклад, код `MainForm->Left=200` змінюватиме значення властивості `Left` головної форми під час виконання програми.

Властивості мають два специфікатори доступу, які використовуються при читанні та модифікації їх значень. Це специфікатор читання (read specifier) та специфікатор запису (write specifier). Специфікатори доступу пов'язують методи читання і запису з властивістю. Так, код `MainForm->Left=200` використовує специфікатор запису, а код `int x=MainForm->Left` використовує специфікатор читання.

Значення властивостей визначаються розробником компонента. Властивість може бути доступна для читання, для читання і запису, для запису або під час виконання.

Деякі властивості (наприклад, властивість `Lines` компонента `Memo`) використовують масиви в якості відповідних членів-даних. Властивості можуть бути екземплярами інших класів бібліотеки візуальних компонентів VCL. Деякі властивості (наприклад, властивість `Style` об'єкту `Font`) представляють собою множини (set) - набори можливих значень властивостей або перелічування (enumerations) - списки можливих значень властивостей.

Методи в компонентах VCL - це функції, які можуть бути викликані для виконання компонентом певних дій. Наприклад, усі візуальні компоненти мають метод `Show()` для відображення

компонента та метод `Hide()` для приховування компонента. Ці методи викликаються як функції-члени класів за допомогою оператора непрямого доступу:

```
MyWindow->Show();
…
MyWindow->Hide();
```

Методи в VCL можуть бути оголошені як відкриті (public), захищені (protected) чи закриті (private). Лише відкриті методи можуть бути викликані користувачами компонента.

Розробник компонента визначає можливість методів приймати аргументи і повертати значення.

Операційна система Windows є середовищем, керованим подіями. До *подій* Windows належать переміщення миші, клік кнопками миші та натискання клавіш на клавіатурі, активізація меню, переміщення вікна, активізація вікна і т.і. Windows повідомляє програму про подію, надсилаючи відповідне повідомлення.

Кожен компонент спроектовано так, щоб реагувати на певні події. Наприклад, компонент `Button` реагуватиме на клік миші. Реакція на подію, яка відноситься до даного компонента, називається опрацюванням події. Події опрацьовуються функціями - опрацьовувачами подій. Декілька компонентів можуть використовувати спільну функцію-опрацьовувач подій. Опрацювання подій в VCL є достатньо простим. Список подій, на які реагує компонент, наведено у вкладці `Events` вікна інспектора об'єктів. Ім'я події служить одночасно її описом. Імена опрацьовувачів подій присвоєні C++ Builder за замовчуванням, але можуть бути змінені програмістом за допомогою інспектора об'єктів. Аргумент `Sender`, що передається опрацьовувачу події, може використовуватись для визначення компонента, який надіслав повідомлення про подію. Подвійний клік на імені опрацьовувача події у вікні інспектора об'єктів викликає його код. Кожен опрацьовувач події містить параметри, необхідні для коректної реакції на подію.

Бібліотека візуальних компонентів VCL була написана мовою Object Pascal і призначалась для Delphi. При створенні C++

Builder розробники фірми Borland взяли цю бібліотеку і адаптували її до нового середовища. Всі об'єкти VCL розміщуються динамічно - коли користувач розміщує компонент на екранну форму, C++ Builder автоматично створює код для його динамічного розміщення. Об'єкти і класи VCL створюються за допомогою оператора new. Більшість компонентів VCL можуть створюватись як на етапі розроблення, так і під час виконання програми. Функції VCL не мають аргументів за замовчуванням, отже, всі аргументи слід вказувати при виклику функції. Класи VCL не підтримують множинного наслідування, тобто не можна створити новий компонент на основі двох або більше існуючих компонентів. Класи в C++ Builder будуть похідними від класу `TObject` лише у випадку, якщо це явно вказуватиметься.

Ієрархія класів VCL, які належать до компонентів, досить складна. На рис.3.2.2 [26] показані деякі базові та похідні класи.

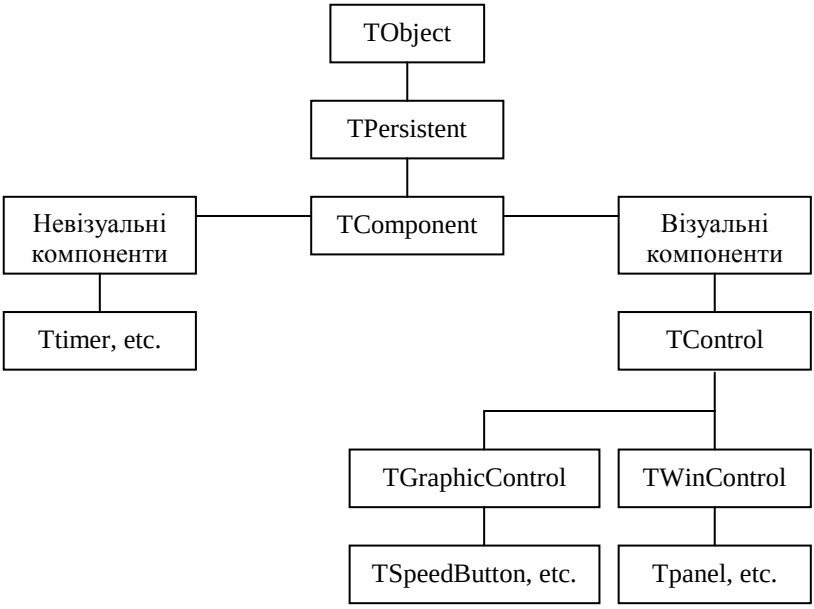

Рис.3.2.2 - Ієрархія класів VCL

Клас TObject є предком усіх класів. Клас TPersistent визначає здатність компонентів зберігати себе в файлах та пам'яті. Клас TComponent також є базовим класом. Він забезпечує функціювання всіх основних компонентів, на його основі створені невізуальні компоненти. Основою візуальних компонентів служить клас TControl, який забезпечує функції, потрібні для візуальних компонентів.

Класи форм і додатків призначені для представлення форм і об'єктів Application в VCL. Всі ці класи є похідними TComponent. Клас TApplication використовується для відображення вікон повідомлень, управління контекстно-залежною довідкою та виведення підписів до кнопок та рядків стану вікон. Клас TForm призначений для інкапсуляції форм для головних, діалогових, дочірніх та будь-яких інших типів вікон.

До класів компонентів входять:

1) класи стандартних компонентів: TButton - кнопка, TEdit, TListBox - список, TMemo, TMainMenu - головне меню додатку, TPopupMenu, TCheckBox, TRadioButton - радіокнопка, TRadioGroup, TGroupBox, TPanel - панель інструментів, TBitBtn - кнопка із зображенням, TSpeedButton - графічне зображення кнопки, TMaskEdit, TStringGrid - виведення інформації у вигляді таблиці, TDrawGrid - виведення інформації у вигляді таблиці, TImage - дозволяє розмістити на формі зображення з файлу, TShape, TBevel - створення рельєфних елементів, TScrollBox;

2) класи спеціальних компонентів управління Win32: TListView, TTreeView, TProgressBar, TTabControl, TPageControl, TRichEdit, TImageList, TStatusBar, TAnimate, TToolBar, TCoolBar;

3) класи компонентів доступу до баз даних: невізуальні - TDataSource, TDatabase, TTable, TQuery, візуальні - TDBGrid, TDBNavigator, TDBText, TDBEdit, TDBListBox, TDBImage;

4) класи стандартних діалогів: TOpenDialog, TSaveDialog, TFontDialog, TColorDialog,

TPrintDialog, TPrinterSetupDialog, TFindDialog, TReplaceDialog;

5) класи системних компонентів: TTimer - системний таймер, TMediaPlayer - програвач аудіозаписів, TPaintBox - порожній "холст" для малювання, а також класи для підтримки OLE і DDE;

6) група Win3.1: TTabSet, TNotebook, TFileListBox, TDirectoryListBox, TDriveComboBox, TFilterComboBox;

7) класи GDI: TCanvas - "холст", TBrush - "пензль", TBitmap - растрові зображення, TFont - операції зі шрифтами;

8) службові класи: TIniFile, TRegistry, TRegKeyInfo, TRect, TPoint, TStringList, TList, TStream, TFileStream, TMemoryStream, TResourceStream.

3.2.2 Інтегроване середовище розроблення C++ Builder

Інтегроване середовище розроблення (IDE) C++Builder складається з: головного меню та панелі інструментів, палітри компонентів, редактора форм, редактора коду, інспектора об'єктів, менеджера проектів.

Палітра компонентів призначена для вибору компонентів або інших елементів керування, які розміщуються на формі. Для розміщення компоненту на форму слід обрати в палітрі компонентів відповідну кнопку, клікнути на формі в тому місці, де повинен розташовуватись лівий верхній кут компонентf. Якщо при виборі компонента натиснути клавішу Shift, то можна буде додати на форму декілька його копій. Якщо на кнопці компонента в палітрі компонентів виконати подвійний клік (double click), то компонент з'явиться в центрі форми (по горизонталі і вертикалі).

Форми є основними складовими додатків C++Builder. Кожен GUI-додаток має принаймні одну форму, яка є головним вікном. Форма головного вікна може бути порожнім вікном, може містити елементи управління або растрове зображення. В типовій програмі форма зазвичай містить меню. Для створення діалогових

вікон в C++ Builder також застосовуються форми. Діалогові вікна відрізняються від звичайних вікон тим, що мають [26]:

1) фіксований розмір;

2) кнопку OK;

3) кнопки Cancel і Help;

4) в рядку заголовка лише кнопку закриття вікна;

5) вкладки;

6) клавішу Tab для переміщення між елементами управління діалогового вікна.

Типове діалогове вікно About представлене на рис. 3.2.3.

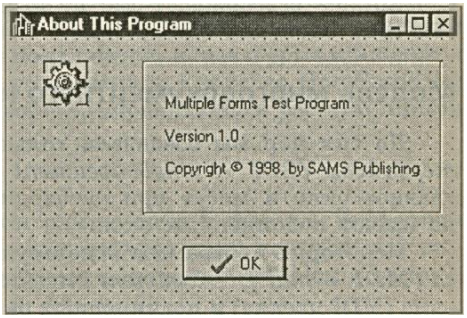

Рис.3.2.3 - Типове діалогове вікно About

Після завершення розроблення діалогового вікна на основі його візуального представлення формується сценарій ресурсів (текстовий файл, який перетворюється в двійковий файл ресурсів за допомогою компілятора ресурсів). Сценарій ресурсів містить інформацію, на основі якої Windows будує діалогове вікно під час виконання програми, - кількість і тип елементів управління у вікні, їх розміри, положення, текст, опції і т.і.

Модальне діалогове вікно повинно бути закрите перед продовженням роботи з додатком. При виклику такого діалогового вікна головне вікно додатку блокується. Прикладом модального діалогового вікна є вікно стану компілятора в C++ Builder. Немодальне діалогове вікно дозволяє користувачу продовжувати роботу з додатком (наприклад, діалогове вікно Find в текстових процесорах). Для створення модального вікна потрібен метод

ShowModal() класу TForm, а для створення немодального вікна - метод Show().

C++ Builder має також стандартні діалогові вікна, представлені класами TOpenDialog, TSaveDialog, TFontDialog і т.і. Вони не є формами C++ Builder, а містяться в системному файлі COMDLG32.DLL, надаються операційною системою і доступні всім додаткам Windows.

Додатки з одним головним вікном, які можуть відображати діалогові вікна, але не мають вікон-нащадків, є додатками з *однодокументним інтерфейсом (SDI-інтерфейсом)*. Деякі програми можуть використовувати модель *багатодокументного інтерфейсу (MDI-інтерфейсу)*. MDI-додатки складаються з головного вікна (предка) та підлеглих вікон (нащадків). Прикладом програм, які використовують модель MDI, є редактор системної конфігурації Windows (SYSEDIT) та менеджер програм Windows. Однією з основних властивостей моделі MDI є те, що вікна-нащадки обмежені головним вікном і можуть переміщатись лише в межах цього вікна. Для створення MDI-додатку в C++ Builder потрібно встановити для властивості FormStyle головної форми значення fsMDIForm. При цьому властивість FormStyle кожного з вікон-нащадків MDI повинна мати значення fsMDIChild.

Основні властивості форм:

1) властивості, доступні під час розроблення і виконання програми - ActiveControl (вибір елемента управління, активного при звертанні до форми); AutoScroll, HorzScrollBar, VertScrollBar (управління лінійками прокрутки); BorderStyle (тип рамки); ClientWidth, ClientHeight (ширина і висота робочої області замість ширини і висоти форми); Font (шрифт, що використовується в межах форми); FormStyle; HelpContext (ідентифікатор контекстної довідки); Icon (значок, що використовується в рядку заголовка форми при виконанні програми, а також для згорнутої форми); Position (розмір і положення форми при першому

відображенні); Visible (початкова видимість форми); WindowState (визначення поточного стану форми);

2) властивості, доступні лише під час виконання програми - ActiveMDIChild (повертає вказівник на активне в даний момент вікно-нащадок), Canvas (поверхня форми, доступна для малювання), ClientRect (координати робочої області форми), Handle (повертає дескриптор вікна для форми), ModalResult (закриття модального вікна), Owner (вказівник на власника форми), Parent (вказівник на породжуючий об'єкт форми).

Перелік основних методів форм наведено у таблиці 3.2.1.

Таблиця 3.2.1 - Перелік основних методів форм

Метод	**Дія методу**
Show(), ShowModal()	Відображення відповідно немодальної та модальної форм
BringToFront()	Розташування форми над усіма формами додатку
Close(), CloseQuery()	Закриття форми після попереднього виклику CloseQuery(), який перевіряє, чи можливе коректне закриття
Print()	Виведення на друк вмісту форми
ScrollInView()	Прокручування форми, доки вказаний компонент не потрапить у поле зору
SetFocus()	Активізація і переміщення форми поверх інших вікон
Методи MDI, застосовні лише до головних вікон MDI	ArrangeIcons() - впорядковує значки всіх згорнутих вікон-нащадків в головному вікні, Cascade() - розташовує каскадом всі незгорнуті вікна-нащадки, Title() - розташовує відкриті вікна-нащадки мозаїкою, Next() - активізує наступне вікно в списку вікон-нащадків, Previous() - активізує попереднє вікно-нащадок

Перелік основних подій форм наведено у таблиці 3.2.2.

Таблиця 3.2.2 - Перелік основних подій форм

Подія	**Момент виникнення**
`OnActivate`	При активуванні форми
`OnClose`	При закритті додатку
`OnCloseQuery`	При перевірці можливості коректного закриття форми
`OnCreate`	При початковому створенні та ініціалізації форми
`OnDestroy`	Звільнення динамічно виділеної пам'яті або інших завершуючих дій, при закритті форми
`OnDragDrop`	При розташуванні об'єкта на форму
`OnMouseDown`, `OnMouseMove`, `OnMouseUp`	Для реакції на кліки миші на формі
`OnPaint`	При необхідності перемалювання форми
`OnResize`	При зміні розміру форми
`OnShow`	Перед тим, як форма стане видимою

При створенні форми генеруються різні події в наступному порядку: конструктор форм, `OnCreate`, `OnShow`, `OnActivate`. При закритті форми події відбуваються в наступному порядку: `OnCloseQuery`, `OnClose`, `OnDestroy`, деструктор форми (якщо він існує).

Інспектор об'єктів є складовою частиною IDE C++ Builder. Він працює сумісно з редактором форм та призначений для встановлення доступних на етапі розроблення властивостей, які визначають поведінку компонента під час виконання програми. Вікно інспектора об'єктів складається з трьох основних частин [26]:

1) селектор компонентів - комбінований список, розташований в верхній частині інспектора, який спадає для вибору компонент для перегляду і редагування (рис.3.2.4). Селектор компонентів відображає ім'я компонента та клас, від якого цей

компонент походить (ім'я класу присутнє лише в верхній частині вікна селектора);

2) вкладка `Properties` - відображає всі властивості обраного компонента, доступні на стадії розроблення. Так, на рис.3.2.1 зображено інспектор об'єктів, який показує властивості компонента `Form1`. Вкладка розділена на 2 стовпці - `Property` (ліворуч) і `Value` (праворуч), які відповідно показують ім'я властивості та значення властивості для обраного компонента;

3) вкладка `Events` - список подій, які може опрацьовувати даний компонент. Для створення опрацьовувача подій достатньо виконати подвійний клік в стовпці `Value` поруч з назвою цієї події. При цьому C++ Builder створить функцію з параметрами, необхідними для опрацювання події. На екрані з'явиться вікно редактора коду з курсором, розташованим всередині опрацьовувача, після чого вводиться код для опрацювання події.

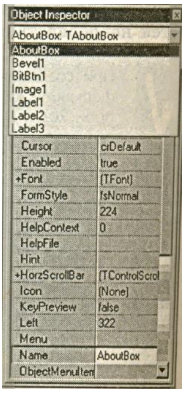

Рис.3.2.4 - Список селектора компонентів

3.2.3. Редактор форм та редактор меню

Редактор форм C++ Builder представляє собою потужний інструмент візуального програмування. Він дозволяє розташовувати, виділяти, пересувати, вирівнювати компоненти, змінювати їх розміри; змінювати розміри та положення самої форми, додавати меню та створювати спеціалізовані діалогові вікна і т.і.

Редактор форм має своє контекстне меню. Пункти контекстного меню вказані в таблиці 3.2.3.

Таблиця 3.2.3 - Пункти контекстного меню редактора форм

Пункт	Опис
Align To Grid	Прив'язка виділених компонентів до сітки редактора форм
Bring To Front	Переміщення виділених компонентів на передній план
Send To Back	Переміщення виділених компонентів на задній план
Revert to Inherited	Повернення формі, взятій з репозиторію об'єктів, початкового вигляду
Align	Відображення діалогового вікна вирівнювання
Size	Відображення діалогового вікна розміру
Scale	Відображення діалогового вікна масштабу
Tab Order	Відображення діалогового вікна зміни порядку перемикання клавішею Tab
Creation Order	Відображення діалогового вікна порядку створення
Add to Repository	Розміщення форми в репозиторій об'єктів для наступного використання
View as Text	Відображення у вікні редактора коду опису форми в текстовому вигляді

Редактор форм містить спеціальну сітку, яка полегшує розроблення форм. За замовчуванням ця сітка розміщена на екрані. Компоненти, які розташовуються на форму, "прив'язуються" до найближчої точки сітки (верхній лівий кут компонента розміщується в цій точці).

Після розміщення компонента на форму потрібно його виділити, щоб змінити якимось чином. Виділення потрібне для переміщення компонента, зміни властивостей компонента, вирівнювання компонента, зміни розміру компонента, копіювання

або переміщення компонента в буфер обміну, поміщення компонента на передній чи задній план, видалення компонента.

Для виділення одиночного компонента слід виконати клік на ньому. На рис.3.2.5 представлено приклад форми з виділеним компонентом `Button`. Властивості та події виділеного компонента відображаються у вікні інспектора об'ктів. Для скасування виділення слід виконати клік на вільній ділянці форми.

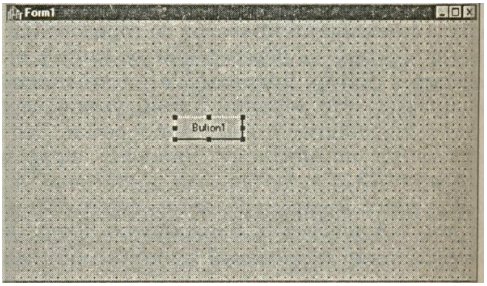

Рис.3.2.5 - Форма з виділеним компонентом Button

Для виділення групи компонентів необхідно здійснити одну з дій:

1) натиснути клавішу Shift + клік миші на потрібних компонентах;

2) натиснути кнопку миші та замкнути компоненти в рамку;

3) обрати в головному меню пункт `Edit|Select All` (виділяє всі компоненти форми).

Переміщення компонентів здійснюється перетягуванням компонента за допомогою курсора миші. Компоненти не можуть бути переміщені за межі компонента-предка. Не можна переміщувати групу компонентів, створених на основі різних базових класів. Для скасування рішення про переміщення компонента достатньо натиснути клавішу Esc перед тим, як відпустити кнопку миші. Положення виділеного компонента коригується за допомогою клавіш зі стрілками на клавіатурі з утриманням натиснутою клавіші Ctrl.

На завершуючому етапі розробки форми варто зафіксувати компоненти на своєму місці, щоб запобігти їх випадковому зміщенню. Для цього слід обрати в головному меню пункт `Edit|Lock Controls`. Положення і розмір компонента буде зафіксовано. Для скасування фіксації слід обрати цей пункт ще раз.

Для досягнення певного візуального ефекту (наприклад, ефекту тіні) компоненти можна накладати один на одного за допомогою пунктів контекстного меню `Send To Back` та `Bring To Front` або таких самих пунктів меню `Edit`.

За допомогою буферу обміну компоненти можна копіювати та переміщувати. Пункт меню `Edit|Copy` дозволяє скопіювати виділений компонент в буфер обміну, `Edit|Cut` - вирізати виділений компонент в буфер обміну, `Edit|Paste` - помістити компонент з буфера обміну на форму або в контейнер для інших компонентів.

Ряд компонентів при розміщенні на формі приймають розмір за замовчуванням. Наприклад, стандартна кнопка має висоту 25 і ширину 75 пікселів. Розмір виділеного компонента змінюється за допомогою миші, діалогового вікна `Size` або за допомогою діалогового вікна `Scale`. Зміна розміру компонента за допомогою миші: встановити курсор миші на один з 8 маркерів компонента, після чого він перетвориться на чорну подвійну стрілку, далі потягнути за маркер, натискаючи кнопку миші. Така зміна розміру можлива лише для візуальних компонентів. Діалогове вікно `Size` (пункт меню `Edit|Size` або пункт `Size` контекстного меню) представлене на рис.3.2.6. За його допомогою можна змінити ширину і висоту всіх компонентів групи. Ще одним інструментом для зміни розміру є діалогове вікно `Scale` (пункт меню `Edit|Scale` або пункт `Scale` контекстного меню), яке дозволяє задати ступінь масштабування у відсотках (рис.3.2.7).

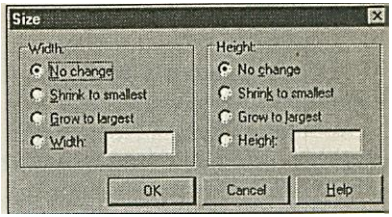

Рис.3.2.6 - Діалогове вікно Size

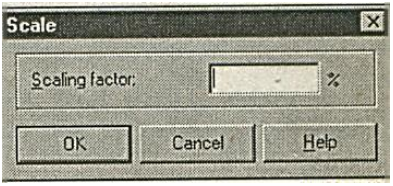

Рис.3.2.7 - Діалогове вікно Scale

Іноді після розташування компонентів на формі виникає потреба вирівнювання компонентів вздовж однієї лінії, центрування відносно форми або розташування з певним кроком. Для вирівнювання компонентів використовують палітру вирівнювання (`Alignment Palette`), діалогове вікно `Alignment` або змінюють значення властивості `Align`. Для відображення палітри вирівнювання слід обрати в головному меню пункт `View|Alignment Palette`. Після цього з'являється палітра вирівнювання з 10 кнопками, кожна з яких відповідає за свій вид вирівнювання (наприклад, вирівнювання центрів по вертикалі або вирівнювання лівих чи правих країв і т.д.). Властивість `Align` визначає спосіб вирівнювання компонента у вікні. Значення властивості `Align` наведені у таблиці 3.2.4.

Таблиця 3.2.4 - Значення властивості `Align`

Значення	**Опис**
(1)	**(2)**
`alBottom`	Компонент розташовується вздовж нижнього краю вікна (наприклад, рядок стану)
`alClient`	Компонент заповнює всю робочу область вікна або всю область вікна, не зайняту іншими компонентами (наприклад, `Memo, Image, RichEdit`)
`alLeft`	Компонент розташовується вздовж лівого краю вікна (наприклад, ліва вертикальна панель інструментів)

(1)	(2)
alNone	Компонент розташовується в довільному місці вікна
alRight	Компонент розташовується вздовж правого краю вікна (наприклад, права вертикальна панель інструментів)
alTop	Компонент розташовується вздовж верхнього краю вікна (наприклад, панель інструментів)

Для створення меню в C++ Builder використовується спеціальний *редактор меню*, який забезпечує можливості:

1) створення головного та контекстних меню;
2) забезпечення оперативного доступу до редактора коду для опрацювання подій OnClick, що відповідають пунктам меню;
3) вставлення меню з шаблонів або файлів ресурсів;
4) збереження меню користувача у вигляді шаблонів.

Всі команди редактора меню доступні через його контекстне меню або через інспектор об'єктів. Контекстне меню редактора показане на рис.3.2.8. Призначення пунктів меню слідує з їх назв.

Головне меню представлене компонентом MainMenu, якому відповідає клас TMainMenu бібліотеки VCL. Кожен пункт меню, в свою чергу, є компонентом MenuItem, інкапсульованим в класі TMenuItem. Для створення пункту меню достатньо розташувати на форму компонент MainMenu і зімнити його значення Name на MainMenu. Після подвійного кліку на значку головного меню на екрані з'явиться редактор меню.

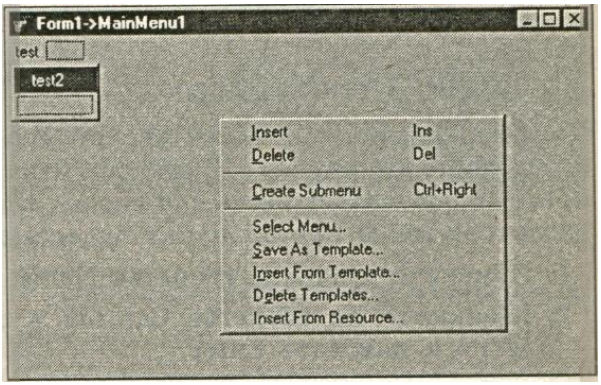

Рис.3.2.8 - Контекстне меню редактора меню

Для створення розділювача меню (горизонтальна лінія в меню, яка розділяє групи пунктів меню) слід перейти до порожнього пункту меню, встановити для властивості `Caption` значення "-" (дефіс) і натиснути Enter.

Для вставлення меню з шаблону потрібно клікнути на порожньому пункті меню, далі клікнути на ньому правою кнопкою миші і обрати у контекстному меню пункт `Insert From Template`, після чого з'явиться вікно `Insert Template` (рис.3.2.9), в якому приведено список доступних шаблонів. Можна використовувати раніше визначені або створювати власні шаблони.

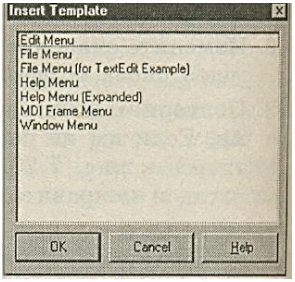

Рис.3.2.9 - Діалогове вікно `Insert Template`

При створенні додатків Windows рідко вдається одержати бажаний результат одразу. Користувачі захочуть додаткових можливостей, керівництво висуне свої вимоги, і можливо ряд

пунктів меню доведеться видаляти. Для видалення пунктів меню, створеного з шаблону, потрібно клікнути на пункті меню, яке буде коригуватись, обрати пункт, який видалятиметься, і натиснути клавішу Delete або обрати пункт меню `Delete` контекстного меню редактора.

Для вставлення пунктів меню потрібно клікнути на тому пункті, над яким слід розмістити новий пункт, і натиснути клавішу Insert або обрати `Insert` в контекстному меню редактора. Після цього можна встановити властивості `Caption` і `Name` для нового пункту.

Пункти меню можна пересувати як у межах одного спливаючого меню, так і між різними меню. Переміщення пунктів можна здійснити двома способами. Перший полягає у використанні буфера обміну, другий - у використанні техніки drag-and-drop.

Для створення підменю слід вибрати у контекстному меню редактора пункт `Create Submenu` або натиснути одночасно Ctrl і стрілку праворуч на клавіатурі.

Для додавання сполучень клавіш, відповідних пунктам меню, потрібно змінити значення властивості `ShortCut` в інспекторі об'єктів.

Після створення всіх необхідних пунктів меню слід написати код, який забезпечить їх функціональність. Для цього потрібно викликати відповідні методи класу `TMemo`. Перш ніж писати код, на формі слід розташувати компоненти `OpenDialog` та `SaveDialog` і вирівняти значки `MainMenu, OpenDialog` та `SaveDialog` вздовж однієї лінії.

Контекстні меню створюються аналогічно головному меню. C++ Builder дозволяє співставити кожному компоненту своє контекстне меню за допомогою властивості `PopupMenu`. При натисканні правої кнопки миші на цьому компоненті на екрані буде автоматично відображатись відповідне меню. Опрацьовувачі для пунктів контекстного меню пишуться аналогічно пунктам головного меню.

C++ Builder надає декілька шаблонів меню, які використовують в головних та контекстних меню. В якості

шаблонів можна зберігати і власностворені меню, щоб надалі використовувати їх в інших програмах. Спочатку потрібно створити нове меню, після чого обрати в контекстному меню редактора пункт `Save As Template`. Це призведе до відкриття діалогового вікна `Save Template`. Далі слід надати створеному меню інформативне ім'я і натиснути кнопку ОК. Меню буде створено як шаблон і відображатиметься разом із стандартними шаблонами C++ Builder в діалоговому вікні `Insert Template`. Для видалення шаблону слід обрати в контекстному меню редактора пункт `Delete Templates`, після чого у діалоговому вікні `Delete Templates` можна буде обрати шаблон, який підлягає видаленню.

3.2.4. Компоненти бібліотеки візуальних компонентів

Компоненти забезпечують потужність C++ Builder. Розглянемо відмінності між компонентами VCL і елементами управління Windows. До елементів управління належать елементи редагування, вікна списків, комбіновані списки, статичні елементи управління (мітки) і кнопки. Елементи управління не мають властивостей, методів та подій. Вони використовують механізм повідомлень, які вказують елементу, що потрібно робити, або передають інформацію від елемента. Компонент VCL - це клас, інкапсулюючий елементи управління Windows. В компоненті VCL до елементів управління додаються властивості, методи і події, які спрощують роботу з ними. VCL пропонує новий підхід до роботи з елементами управління: усі компоненти VCL - це елементи управління, але не всі елементи управління є компонентами.

До візуальних компонентів належать елементи редагування, кнопки, списки, мітки і т.і. При розробленні програми візуальні компоненти виглядають такими, як вони будуть виглядати під час її виконання. Візуальні компоненти користувач бачить під час розроблення програми. Невізуальні компоненти є невидимими в процесі розробки програми. Прикладами невізуальних компонентів є системні таймери, компоненти для роботи з базами даних, списки зображень.

Властивість `Name` має для компонентів важливе значення. При розміщенні компонента на формі C++ Builder створює вказівник на компонент і використовує властивість `Name` в якості імені змінної та для утворення імен опрацьовувачів подій. Для кожного розташованого на формі компонента C++ Builder встановлює значення властивості `Name` за замовчуванням. Але імена за замовчуванням можна залишити лише для тих компонентів, до яких ніколи не буде звертань у програмі. Властивість `Name` можна змінювати в будь-який час за допомогою інспектора об'єктів. Після зміни властивості `Name` певного компонента C++ Builder вносить відповідні зміни в код, згенерований ним автоматично, але залишає без змін текст, написаний програмістом. Ні в якому разі не можна змінювати властивість `Name` під час виконання програми. Не можна змінювати вручну ім'я компонента, яке C++ Builder присвоює вказівнику на компонент, або імена опрацьовувачів подій у вікні редактора коду.

Властивість `Color` визначає колір фону компонента. Колір тексту встановлюється властивістю `Font`. Порядок роботи з властивістю `Color` в інспекторі об'єктів дещо унікальний. Якщо клікнути в стовпці `Value`, то з'явиться кнопка зі стрілкою вниз, яка вказує на можливість вибору кольору зі списку. Крім цієї можливості є ще одна - після подвійного кліку в тому ж стовпці `Value` на екрані з'явиться діалогове вікно `Color` (рис.3.2.10), в якому можна обрати один з раніше визначених кольорів або створити власні кольори при натисненні кнопки `Define Custom Colors`. Те ж діалогове вікно `Color` відображатиметься при використанні в додатку компонента `ColorDialog`.

Кольори слід використовувати для акцентування уваги і виділення. По можливості у програмі слід використовувати системні кольори. Більш детальні рекомендації щодо використання кольору приведені в пункті 1.3.4 розділу 1.

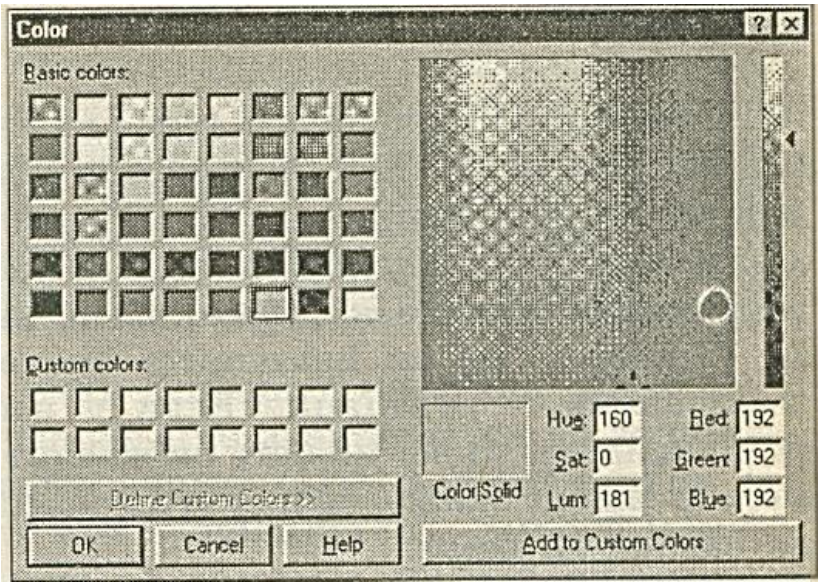

Рис.3.2.10 - Діалогове вікно Color

Властивість `Cursor` визначає вигляд курсору миші під час його пересування по компоненту. Для деяких компонентів Windows змінює курсор автоматично. Наприклад, коли курсор розташований над компонентами `Edit`, `Memo` або `RichEdit`, він змінюється на вертикальний штрих "I". Для певних вікон можна визначити спеціальний курсор, наприклад, для позначення тривалого процесу краще використовувати курсор у вигляді пісочного годинника (значення `crHourGlass`).

Властивість `DragCursor` визначає курсор, який використовується при перетягуванні компонента в тому випадку, коли адресат може прийняти об'єкт, що перетягується.

Властивість `Enabled` дозволяє або забороняє доступ до компонентів. Коли компонент недоступний, він не може одержувати фокус введення (клік на такому об'єкті не дає жодного ефекту). Для позначення такого компоненту, як правило, застосовується деякий візуальний ефект. Наприклад, для позначення недоступної кнопки її текст виділяється сірим

кольором. Властивість `Enabled` має логічний тип: `true` - компонент доступний, `false` - компонент недоступний.

Властивість `Font` - одна з основних. Це екземпляр класу `TFont`, який має власні властивості. Властивості для `Font` можна визначити після подвійного кліку на імені шрифта в інспекторі об'єктів або після виклику діалогового вікна `Font`. На рис.3.2.11 зображено вікно інспектора об'єктів з властивістю `Font`, відкритою для відображення властивостей класу `TFont`. Властивість `Color` визначає колір шрифта, `Name` дозволяє обрати гарнітуру шрифта (гарнітура шрифта - набір шрифтів, які відрізняються за розміром, нарисом, співвідношенням розміру висоти великих та малих літер, густотою, але близькі за характером та розпізнавальними знаками рисунку [27]), `Height` використовується для визначення висоти шрифта в пікселях, `Size` визначає висоту шрифта в пунктах. Властивість `Style` може використовуватись для встановлення напівжирного, курсивного підкресленого або перекресленого шрифтів.

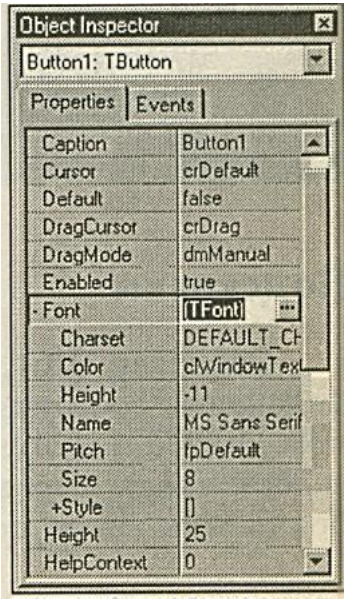

Рис.3.2.11 - Інспектор об'єктів з відкритою властивістю `Font`

Властивість `Hint` задає текст підказки для компонента. Текст підказки складається з двох частин. Першу частину називають короткою підказкою - саме цей текст виводиться, коли користувач затримує курсор над компонентом. Спливаюче вікно, в якому відображається текст підказки, називають спливаючою підказкою. Другу частину тексту підказки називають довгою підказкою - це необов'язковий текст, який виводиться в рядку стану, коли користувач встановлює курсор миші на компоненті. Тексти короткої та довгої підказок відокремлюється один від одного символом "|".

Властивості `ParentColor`, `ParentCtl3D`, `ParentFont i ParentShowHint` працюють однаково. Коли ці властивості мають значення true, компонент наслідує властивості `Color, Ctl3D, Font i ShowHint` від свого предка.

Властивість `Tag` представляє собою зарезервовану змінну розміром в 4 байти. Цю властивість можна використовувати для зберігання даних, потрібних для роботи компонента, наприклад, вказівника на інший клас, індексного значення і т.і.

Ряд часто використовуваних властивостей перераховано в таблиці 3.2.5 [26].

Таблиця 3.2.5 - Додаткові властивості компонентів

Властивість	**Опис**
(1)	**(2)**
`BorderStyle`	Приймає значення `bsSingle` або `bsNone` (злиття компоненту з фоном)
`Caption`	Задає напис на компоненті, якщо це потрібно
`Ctl3D`	Вказує, чи повинен елемент управління виділятись об'ємною рамкою
`Height`	Визначає висоту компонента
`HelpContext`	Використовується для зв'язування індексу в довідковому файлі з компонентом
`Left`	Визначає x-координату аргумента

(1)	(2)
PopupMenu	Визначає контекстне (спливаюче) меню, яке відображається при натисканні правої кнопки миші
TabOrder	Застосовується для компонентів віконного типу для встановлення черги цього компонента при перемиканні клавішею Tab
TabStop	Застосовується для компонентів віконного типу. Вказує, що на даний компонент можна перемкнутись клавішею Tab.
Top	Визначає у-координату компонента
Visible	При читанні вказує, чи є компонент в даний час видимим. При записі значення в Visible компонент може бути зроблено видимим або прихованим
Width	Визначає ширину компонента

Основні спільні методи компонентів представлено в таблиці 3.2.6.

Таблиця 3.2.6 - Спільні методи компонентів

Метод	Опис
(1)	(2)
BroadCast	Використовується для відправлення повідомлень всім компонентам-нащадкам віконного типу
ClientToScreen	Перетворює локальні координати вікна в екранні координати
ContainsControl	Повертає true, якщо вказаний компонент є нащадком компонента або форми
HandleAllocated	Повертає true, якщо для компонента було визначено властивість Handle
Invalidate	Запитує перемальовування компонента при першій можливості

(1)	(2)
Perform	Дозволяє компоненту надіслати повідомлення одразу самому собі, не використовуючи системи повідомлень Windows
Refresh	Запитує негайне перемальовування вікна, попередньо стираючи зображення компонента
Repaint	Запитує негайне перемальовування вікна без попереднього стирання фону компонента
Hide	Приховує компонент
SetBounds	Дозволяє задати властивості Top, Left, Width, Height одночасно
SetFocus	Встановлює фокус введення, робить компонент активним. Застосовується лише для компонентів віконного типу
Update	Викликає негайне примусове перемальовування компонента

Найбільш часто використовувані події компонентів вказано в таблиці 3.2.7.

Таблиця 3.2.7 - Основні події компонентів

Подія	Опис
(1)	(2)
OnChange	Виникає, коли в елементі управління відбуваються якісь зміни
OnClick	Відбувається при кліку на компоненті будь-якою кнопкою миші
OnDblClick	Відбувається при подвійному кліку на компоненті
OnEnter	Відбувається, коли компонент віконного типу одержує фокус введення (активізується)

(1)	(2)
OnExit	Відбувається, коли компонент віконного типу втрачає фокус в результаті перемикання на інший елемент управління
OnKeyDown	Відбувається при натисканні користувачем клавіші, якщо компонент знаходиться в фокусі
OnKeyPress	Відбувається при натисканні однієї з алфавітно-цифрових клавіш, клавіш Tab, Backspace, Enter, Esc
OnKeyUp	Відбувається кожного разу, коли клавішу відпускають
OnMouseDown	Виникає, коли на компоненті натискається кнопка миші
OnMouseMove	Відбувається при переміщенні курсору миші через елемент управління
OnMouseUp	Виникає при відпусканні кнопки миші на елементі управління, якщо перед цим кнопка була натиснута на цьому ж елементі
OnPaint	Виникає кожного разу, коли необхідно перемалювати компонент

Клас TStrings - цей клас використовується для роботи з масивами рядків. Деякі компоненти VCL використовують екземпляри класу TStrings для роботи зі своїми даними (текстовими).

В C++ Builder є чотири стандартних *компоненти для управління редагуванням*:

1) Edit - інкапсулює базовий однорядковий елемент редагування;

2) Memo - інкапуслює багаторядковий елемент редагування;

3) MaskEdit - представляє собою компонент Edit, до якого додано фільтр, або маска даних, що вводяться;

4) RichEdit - дозволяє змінювати шрифти, використовувати відступи, задавати напівжирний, курсивний або

підкреслений текст та інше. Цей компонент є маленьким текстовим процесором.

Компоненти `ListBox` *і* `ComboBox`. Компонент `ListBox` є стандартним елементом управління Windows - вікно списку, яке містить набір елементів, з яких користувач може зробити вибір. Компонент `ComboBox` дозволяє створити комбінований список, цей компонент є комбінацією вікна списку та однорядкового елементу редагування. Користувач може вибрати значення зі списку або ввести потрібне значення в поле редагування.

Бібліотека VCL містить декілька типів *кнопок*, які можна використовувати в своїх додатках. Компоненти кнопок мають чотири основних властивості:

1) `ModalResult` - використовується для закриття форм, відображуваних методом `ShowModal()`;

2) `Default` - якщо фокус введення з клавіатури знаходиться в елементі управління, який не є кнопкою, і користувач натиснув на клавіатурі Enter, діалогове вікно поводитиметься так, неначе користувач натиснув кнопку за замовчуванням, визначену за допомогою властивості `Default`;

3) `Cancel` - працює з клавішею Esc аналогічно, як властивість `Default` працює з клавішею Enter;

4) `Enabled` - використовується для ввімкнення або вимкнення кнопки в залежності від поточного стану програми або окремої форми.

Основні компоненти кнопок представлені в таблиці 3.2.8.

Таблиця 3.2.8 - Компоненти кнопок

Компонент	**Опис**
(1)	**(2)**
`Button`	Стандартна кнопка
`BitBtn`	До стандартної кнопки додано можливість виведення на її поверхню растрового зображення
`RadioButton`	Радіокнопка - набір опцій, з яких може бути обрана лише одна

(1)	(2)
SpeedButton	Кнопка швидкого доступу. Не є компонентом віконного типу, на відміну від компонентів Button та BitBtn, тому не може одержати фокус введення, і на неї не можна перемкнутись за допомогою клавіші Tab
CheckBox	Прапорець - використовується для надання користувачу можливості ввімкнути або вимкнути деякий режим або для індикації ввімкненого режиму. Може мати три стани: встановлений, відмінений або заблокований

Компонент `Label` використовується для відображення тексту на формі. Іноді текст мітки визначається лише під час розроблення і надалі не змінюється. В інших випадках мітка є динамічною і змінюється під час виконання програми. Для вказання тексту мітки використовується властивість `Caption`.

Компонент `ScrollBar` - це автономна лінійка прокрутки, не пов'язана з елементом редагування, списком або формою.

Компонент `Panel` використовується для розміщення кнопок швидкого доступу та звичайних кнопок, відображення текстових міток (наприклад, заголовку форми) та графічних зображень. Однією з переваг панелі є те, що компоненти, розташовані на панелі, стають її нащадками і завжди слідують за панеллю, куди б вона не пересувалась.

Windows містить цілий набір *стандартних діалогових вікон*, які може використовувати будь-який додаток. До них належать:

1) `File Open` - вікно відкриття файлу в додатку;

2) `File Save` - вікно збереження файлу під певним іменем;

3) `File Open Picture` - діалогове вікно відкриття файлу з доданим вікном перегляду;

4) `File Save Picture` - діалогове вікно збереження документу з доданим вікном перегляду;

5) `Font` - вікно вибору шрифта з усіх доступних в системі шрифтів;

6) `Color` - вікно вибору кольору;

7) `Print` - вікно друку;

8) `Print Setup` - вікно встановлення параметрів друку;

9) `Find` - вікно пошуку тексту;

10) `Replace` - вікно заміни тексту.

Стандартні діалогові вікна знаходяться на вкладці `Dialogs` палітри компонентів. Ці компоненти належать до невізуальних, оскільки не мають візуального інтерфейсу на етапі розроблення програми. Для створення і відображення усіх стандартних діалогових вікон використовується метод `Execute()`, який повертає значення `true`, якщо користувач натиснув кнопку ОК, двічі клікнув на імені файлу або натиснув клавішу Enter. Якщо користувач натиснув кнопку Cancel, натиснув клавішу Esc або закрив діалогове вікно системною кнопкою закриття вікна, то `Execute()` повертає значення `false`.

3.3 Засоби розроблення інтерфейсів користувача у середовищі Delphi

3.3.1 Компоненти інтерфейсу користувача в Delphi. Робота з графікою та файлами в Delphi

На базовому рівні інтерфейс користувача засновується на чотирьох класах Delphi:

1) `TApplication` - фундаментальний клас, властивості і методи якого описують основні характеристики додатків Windows;

2) `TClipboard` - визначає взаємодію з буфером обміну;

3) `TForm` - клас форми, описує властивості і методи для створення вікон, "будівельних блоків" для розташування візуальних та невізуальних компонентів; всі його методи і властивості подібні до методів і властивостей класу `TForm` мови C++ Builder;

4) `TScreen` - засоби для задавання системних налагоджень: розрішення екрану, доступних шрифтів і т.і.

На основі *класу* *`TApplication`* програміст може опрацьовувати системні повідомлення Windows, призначені для додатку, організовувати оперативні підказки, виконувати інші специфічні дії, які залежать від операційної системи. Кожна програма, створювана в Delphi, має доступ до глобальної змінної `Application` класу `TApplication`. До неї слід звертатись, коли необхідно скористатись можливостями цього класу. Основні властивості, методи і події класу `TApplication` [28] наведені в таблицях 3.3.1-3.3.3.

Таблиця 3.3.1 - Основні властивості класу `TApplication`

Властивість	**Призначення**
(1)	**(2)**
`Active`	Має значення true, якщо додаток активний (має фокус введення)
`ExeName`	Визначає повне ім'я файлу програми разом зі шляхом пошуку
`CurrentHelpFile`	Ім'я поточного довідкового файлу

(1)	(2)
Handle	Внутрішній ідентифікатор програми в системі Windows
HelpFile	Ім'я довідкового файлу, який використовується за замовчуванням
Hint	Текст спливаючої підказки за замовчуванням
HintColor	Колір вікна спливаючої підказки
HintHidePause	Часовий інтервал в мілісекундах, по проходженні якого спливаюча підказка буде прихована
HintPause	Часовий інтервал в мілісекундах, по проходженні якого спливаюча підказка з'явиться на екрані
HintShortPause	Часовий інтервал в мілісекундах, по проходженні якого спливаюча підказка з'явиться на екрані, для випадку, коли вже відображається інша підказка
HintShortCuts	Має значення true, якщо в тексті спливаючої підказки додатково відображатиметься інформація про "гарячу клавішу"
Icon	Значок (клас TIcon), який використовується системою Windows для ідентифікації даного додатку
MainForm	Властивість має тип TForm і визначає головну форму програми
ShowMainForm	Має значення true, якщо головною вважається головна форма етапу проектування
Terminated	Має значення true, якщо додаток одержав від Windows повідомлення WM_QUIT, яке означає завершення роботи додатку
Title	Заголовок додатку - рядок, який відображається на кнопці Панелі задач

Таблиця 3.3.2 - Основні методи класу `TApplication`

Метод	**Призначення**
(1)	**(2)**
`Procedure ActivateHint (Cursor Pos: TPoint)`	Відображає спливаючу підказку в заданій точці екрану
`Procedure BringToFront;`	Переміщує останнє з активних вікон на передній план екрану
`Procedure CancelHint;`	Приховує спливаючу підказу
`Procedure HandleMessage;`	Перериває роботу програми для опрацювання чергового системного повідомлення Windows, яке зберігається у черзі повідомлень
`Function HelpCommand (Command:Word; Data:Longint):Boolean;`	Швидкий доступ до системної функції Windows, яка визначається параметром Command і відповідає за роботу довідкової системи
`Function HelpContext (Context:THelpContext): Boolean; Function HelpJump (const JumpID:String):Boolean;`	Відображає вказаний розділ довідкової системи
`Function IsRightToLeft: Boolean;`	Повертає значення true, якщо у додатку використовується робота елементів управління в режимі "зправа наліво"
`Function MessageBox (const Text, Caption:PChar; Flags:Longint):Integer;`	Показує стандартне діалогове вікно, яке містить кнопки, що визначаються параметром `Flags`
`Procedure NormalizeAllTopMosts;`	Переводить всі вікна додатку зі стану "завжди зверху" в звичайний режим
`Procedure Minimize;`	Згортає всі вікна додатку

(1)	(2)
`Procedure Restore;`	Відновлює всі згорнуті вікна програми до початкового розміру
`Procedure NormalizeTopMosts;`	Переводить всі вікна додатку, крім головної форми, зі стану "завжди зверху" в звичайний режим
`Procedure ProcessMessages;`	Працює аналогічно методу `HandleMessage`, але опрацьовує не одне, а всі системні повідомлення з черги
`Procedure RestoreTopMosts;`	Переводить всі вікна з початкового стану в стан "завжди зверху", якщо ці вікна раніше вже знаходились в такому стані
`Procedure ShowException (E:Exceprion);`	Виводить діалогове вікно з повідомленням, яке описує виключення (параметр E)
`Procedure Terminate;`	Завершує роботу додатка
`Function UseRightToLeftAlignment : Boolean;`	Повертає значення true, якщо режим роботи "справа наліво" використовується для вирівнювання об'єктів
`Function UseRightToLeftReading : Boolean;`	Повертає значення true, якщо режим роботи "зправа наліво" використовується для виведення текстової інформації
`Function UseRightToLeftScrollBar : Boolean;`	Повертає значення true, якщо режим роботи "справа наліво" використовується для відображення смуг прокрутки ліворуч від елементів управління

Таблиця 3.3.3 - Основні події класу `TApplication`

Подія	**Умова генерації**
`OnActionExecute` `OnActionUpdate`	Події,не визначені розробником в списку подій
`OnActivate`	Додаток стає активним
`OnDeactivate`	Додаток стає неактивним
`OnException`	Виникає виключна ситуація, яка програмно не контролюється
`OnHelp`	Додаток одержує запит на видачу довідкової інформації
`OnHint`	Користувач перемістив вказівник на елемент управління, здатний видавати спливаючу підказку
`OnIdle`	Додаток знаходиться в стані очікування, не виконуючи програмного коду
`OnMessage`	Додаток одержав системне повідомлення від Windows
`OnMinimize`	Додаток згорнуто
`OnRestore`	Додаток відновлено до початкового розміру із згорнутого стану
`OnShortCut`	Користувач натиснув кнопку
`OnShowHint`	Додаток готується вивести спливаючу підказку

Клас `TClipboard` дозволяє працювати зі стандартним буфером обміну Windows. Він описаний в модулі `Clipbrd`. Спеціально створювати об'єкт типу `TClipboard` не потрібно. В Delphi є готовий об'єкт `Clipboard`, до якого можна звертатись для використання буферу обміну. Властивості і методи класу `TClipboard` [28] наведені в таблицях 3.3.4, 3.3.5. Деякі стандартні формати даних для буферу обміну Windows описуються значеннями-константами, вказаними в таблиці 3.3.6.

Таблиця 3.3.4 - Властивості класу `TClipboard`

Властивість	Призначення
`AsText`	Вміст буферу обміну (БО) у вигляді рядка. Дана властивість також дозволяє заносити інформацію в буфер за допомогою оператора присвоєння
`FormatCount`	Кількість форматів даних БО, які підтримуються
`Formats`	Масив доступних (зареєстрованих) форматів для буферу обміну

Таблиця 3.3.5 - Методи класу `TClipboard`

Метод	Призначення
(1)	**(2)**
`Procedure Assign (Source: TPersistent);`	Копіювання об'єкта, вказаного в якості параметру, в буфер обміну
`Procedure Clear;`	Видалення поточного вмісту буферу обміну
`Procedure Close;`	Закриття буферу обміну. Виконується, коли запис даних у буфер відбувається в декілька прийомів, і чергове занесення інформації не повинне стирати попереднього вмісту буферу обміну. Після закриття буферу обміну запис в нього нових даних знищить стару інформацію
`Function GetComponent (Owner, Parent:TComponent): TComponent;`	Виконання вставки з буферу обміну. Параметр `Owner` визначає об'єкт, який приймає вміст буферу, а параметр `Parent` визначає предка об'єкта `Owner`
`Procedure SetComponent (Component: TComponent);`	Запис даних в буфер обміну

(1)	(2)
`Function GetTextBuf (Buffer:PChar; BufSize:Integer): Integer;`	Одержання даних з буферу обміну в програмний буфер (вказівник `Buffer`)
`Procedure SetTextBuf (Buffer:PChar);`	Запис даних з програмного буферу `Buffer` в буфер обміну
`Procedure HasFormat (Format:Word): Boolean;`	Повертає значення true, якщо дані у буфері обміну зберігаються в форматі, описаному параметром `Format`
`Procedure Open;`	Відкриття буферу обміну. Використовується, коли дані записуватимуться в буфер обміну в декілька прийомів

Таблиця 3.3.6 - Константи, які описують деякі формати даних для буферу обміну Windows

Значення	Формат
`CF_TEXT`	Текст, розділений символами нового рядка (стандартний текстовий формат)
`CF_BITMAP`	Формат точкового зображення Windows
`CF_METAFILEPICT`	Формат графічного метафайлу Windows
`CF_PICTURE`	Об'єкт типу `TPicture`
`CF_COMPONENT`	Інший стандартний об'єкт

За допомогою *класу* `TScreen` можна визначити, які форми є в додатку, дізнатися, яка форма і який елемент управління є активними, одержати відомості про доступні шрифти, вказівники миші. В Delphi є глобальна змінна `Screen` - об'єкт цього класу. Слід звертатись саме до неї, а не створювати екземпляри `TScreen` самостійно.

Основні властивості, методи і події класу `TScreen` [28] наведені в таблицях 3.3.7-3.3.9.

Таблиця 3.3.7 - Основні властивості класу TScreen

Властивість	**Призначення**
(1)	**(2)**
ActiveControl	Елемент управління, який має фокус введення
ActiveForm	Форма, яка має фокус введення
Cursor	Поточна форма вказівника (тип TCursor), використовуваного в додатку
Cursors	Масив вказівників, доступних в додатку
FormCount	Кількість форм програми, відображених на екрані
Forms	Масив видимих форм програми, за допомогою якого можна змінювати будь-які властивості цих форм
DataModuleCount	Кількість модулів даних (спеціальних типів форм)
DataModules	Масив модулів даних
DesktopWidth	Ширина Робочого столу в пікселях
DesktopHeight	Висота Робочого столу в пікселях
DesktopLeft	Ліва координата Робочого столу
DesktopTop	Верхня координата Робочого столу
Fonts	Масив імен шрифтів (тип TStrings), призначених для виведення тексту на екран (в цей масив не потрапляють шрифти, призначені для роботи з принтером)
Width	Ширина екранного простору в пікселях
Height	Висота екранного простору в пікселях
HintFont	Шрифт, яким відображається спливаюча підказка
IconFont	Шрифт, яким виконуються підписи під значками в діалоговому вікні вибору файлів
MenuFont	Шрифт, який використовується для написів в пунктах меню

(1)	(2)
MonitorCount	Кількість моніторів, використовуваних для представлення Робочого столу
Monitors	Масив моніторів (клас TMonitor), доступних в системі
PixelsPerInch	Кількість пікселів на один дюйм екрану монітора (розрішення екрану)

Таблиця 3.3.8 - Основні методи класу TScreen

Метод	Призначення
Procedure EnableAlign;	Дозволяє вирівнювання форм за розмірами екрану (в залежності від значення їх властивості Align)
Procedure DisableAlign;	Забороняє вирівнювання форм за розмірами екрану (в залежності від значення їх властивості Align)
Procedure Realign;	Перевпорядковує форми на екрані в залежності від значення Align
Procedure ResetFonts;	Оновлює список поточних шрифтів

Таблиця 3.3.9 - Основні події класу TScreen

Подія	Умова генерації
OnActiveControlChange	Фокус переміщується на новий елемент управління в поточній формі
OnActiveFormChange	Фокус переміщується на нове вікно (форму) програми

Робота з графікою. Вся технологія виведення графічної інформації на екран заснована на понятті "холста" (класу TCanvas). Цей клас має всі можливості для відображення такої інформації. На його основі створено більшість компонентів Delphi, які є елементами управління і повинні відображатись на екрані з використанням засобів цього класу. За замовчуванням область

"холста" співпадає з клієнтською областю форми або елемента управління. Ця область є тією частиною об'єкта, яка не зайнята допоміжними деталями оформлення, недоступними для виведення на них графічної інформації (наприклад, заголовком вікна, рядком меню, панеллю командних кнопок, границями об'єкту). Клас `TCanvas` має набір стандартних властивостей та методів, які дозволяють виконувати найпростіші графічні операції. Робота більшості цих методів заснована на понятті графічного курсору - видимої позначки на поверхні візуалізації, що позначає місце, де відбувається дія [29] (наприклад, місце, з якого буде малюватись лінія).

Після того, як на клієнтську область форми виведено графічну інформацію, вона відображається в рамках форми, допоки ця область не закриється іншим вікном або додаток не буде згорнуто. При новій появі цієї області на екрані графічні дані, виведені на неї раніше, не відновляться, і їх доведеться виводити знову. Коли клієнтська область форми стає видимою, Windows генерує системне повідомлення `WM_PAINT`, яке описує ту частину форми, яка вимагає перемалювання. В Delphi таке повідомлення позначається `OnPaint`. Воно автоматично опрацьовується формами і елементами управління при необхідності їх відображення і, зазвичай, не вимагає втручання з боку програміста. Виключенням є ті ситуації, коли форма використовується для виведення графічної інформації: графіків, мультимедійних даних і т.і.

Клас `TGraphics` є абстрактним. На його основі створені класи, призначені для використання в програмах графічних об'єктів, наприклад, точкове зображення, значок. Від класу `TGraphics` такі об'єкти наслідують властивості, наведені в таблиці 3.3.10. Методи класу `TGraphics` мають характеристики `virtual` і `abstract` та визначаються у конкретних класах-нащадках. Вони наведені в таблиці 3.3.11.

Таблиця 3.3.10 - Наслідувані властивості класу TGraphics

Властивість	Призначення
Width	Ширина об'єкту в пікселях
Height	Висота об'єкту в пікселях
Modified	Має значення true, якщо об'єкт було змінено
Palette	Ідентифікатор кольорової палітри Windows
Transparent	Має значення true, якщо об'єкт буде малюватись в "прозорому" режимі. Колір, який визначає рівень прозорості, задається в конкретному класі

Таблиця 3.3.11 - Абстрактні методи класу TGraphics

Метод	Призначення
Procedure LoadFromFile (const FileName:string);	Завантаження графічної інформації з файлу
Procedure SaveToFile (const FileName:string);	Збереження графічної інформації у файлі
Procedure LoadFromClipboardFormat;	Завантаження графічної інформації з буферу обміну
Procedure SaveToClipboardFormat;	Збереження графічної інформації у буфері обміну
Procedure LoadFromStream (Stream: TStream);	Завантаження графічної інформації з потоку
Procedure SaveToStream (Stream:TStream);	Збереження графічної інформації у потоці

В якості властивостей, в першу чергу, використовуються класи, які описують колір і спосіб заповнення областей форми, колір і товщину ліній, стиль і розмір шрифта та інші. Додаткові методи призначені для виведення на екран зображень та рисунків.

Клас TPen (Олівець) - визначається властивостями Color (колір олівця), Mode (режим малювання), Style (стиль лінії), Width (товщина лінії в пікселях).

Клас TBrush (Пензль) - призначений для заповнення суцільних областей клієнтської чатини форми відповідно до заданого шаблону. Крім властивостей Color і Style, які

співпадають з аналогічними властивостями класу TPen, в клас TBrush додано нову властивість Bitmap, яка дозволяє заповняти область не лише суцільним кольором або пунктирними лініями, але й наперед підготовленим точковим зображенням.

Клас TFont (Шрифт) - є оболонкою ресурсу Windows, який визначає поточний шрифт. Містить стандартні властивості, які описують характеристики шрифта. Основні - Color (колір), Charset (набір символів, які визначаються використовуваним кодуванням), Height (висота шрифта в пікселях), Name (назва шрифта, під якою він зареєстрований у Windows), Pitch (профіль шрифта, який визначає, буде відстань між символами фіксованою або змінною), Size (висота шрифта в пікселях), Style (стиль шрифта).

Вище були перераховані найважливіші властивості клієнтської області форми, які активно використовуються для виведення графічної інформації. Окрім них необхідно відзначити деякі властивості самого класу TCanvas, наведені у таблиці 3.3.12. Клас TCanvas містить велику кількість методів.

Таблиця 3.3.12 - Деякі властивості класу TCanvas

Властивість	**Призначення**
CanvasOrientation	Властивість, доступна лише для читання, визначає позицію початку координат
ClipRect	Область, яка реально відображується - зазвичай співпадає з клієнтською областю, що означає виведення всієї графічної інформації
CopyMode	Режим копіювання графічного образу на клієнтську область форми
PenPos	Поточна позиція графічного курсору
Pixels	Двовимірний масив, який зберігає кольори кожного пікселя зображення
TextFlags	Спосіб виведення тексту на клієнтську область форми

Нащадки класу TGraphics:

1) клас TBitmap (Точкове зображення) - це спеціальний клас, за допомогою якого можна зберігати, завантажувати з файлу або буферу обміну, зберігати в файлі або буфері обміну графічні точкові зображення у форматі бітової карти (розширення .bmp), а також виконувати над ними ряд допоміжних операцій. Використання цього класу нерозривно пов'язане з поняттям "холста";

2) клас TIcon (Значок) - призначений для роботи з зображеннями в форматі значка Windows (розширення .ico). Його властивості і методи не відрізняються від властивостей і методів класу TBitmap за виключенням того, що значок завжди має певний прозорий колір, а масштабувати його не можливо;

3) клас TMetaFile (Метафайл) - призначений для роботи зі спеціальним типом графічних даних - метафайлом (розширення .emf, .wmf), який відрізняється від точкового зображення тим, що зберігає не пікселі, а спеціальний код, який при виведенні інтерпретується як набір команд;

4) клас TJPEGImage (Зображення в форматі JPG) - призначений для роботи з зображеннями JPEG, представленими в спеціальному форматі, який дозволяє компактно зберігати великі рисунки. При роботі з цим класом малювати на "холсті" не можна, він застосовується лише для відображення на екрані.

Компонент TPaintBox (Область малювання) розташований на панелі System. Він використовується для виділення на формі декількох областей для малювання. Такий підхід зручний, коли в програмі відбувається активне виведення графічної інформації на екран і бажано розділити цей процес на незалежні частини. Цей компонент може охоплювати довільну прямокутну область форми. Він містить єдину головну властивість Canvas, яка має власну систему координат. Єдина оброблювана подія OnPaint генерується автоматично при необхідності перемалювати одну чи декілька областей (або їх частини). Розробник повинен визначити, що буде зображено у кожному об'єкті класу TPaintBox. Цей компонент також може самостійно

викликати перемальовування клієнтської області форми за допомогою методу `Paint`.

Робота з файлами. При роботі з файлами в Delphi можливі два принципово різних підходи. Перший полягає у використанні стандартних підпрограм, які дозволяють записувати вміст змінних у файли і зчитувати їх назад з файлу у змінні. До цих засобів додано бібліотеки стандартних функцій по роботі з файлами, котрі засновані на системних функціях Windows. У зв'язку з появою версії Object Pascal і реалізації поняття класу в мові з'явились засоби об'єктної роботи з даними. Це другий підхід до роботи з файлами в Delphi.

Незалежно від використовуваного підходу технологія роботи з файлами в Delphi вимагає певного порядку дій:

1) файл потрібно відкрити. Це означає, що операційна система "дасть добро" на внесення змін до даного файлу і простежить, щоб звертання інших користувачів і програм до цього файлу виконувались коректно. При відкритті файлу системі управління файлами повідомляється, в якому режимі файл буде відкрито: планується вносити до нього зміни чи файл відкривається лише для зчитування даних. Зазвичай вказується також, яка внутрішня структура файлу, що відкривається. Після того, як файл успішно відкрито, в програму повертається його ідентифікатор - змінна, яка буде використовуватись для ідентифікації цього файлу в усіх процедурах обробки;

2) починається робота з файлом - зчитування з нього даних, запис, пошук та інші операції;

3) файл закривається. Він знову стає доступним іншим додаткам. Закриття файлу гарантує, що всі внесені в нього зміни не зникнуть.

Стандартний підхід до роботи з файлами розглядався в дисципліні "Програмування", тому не є предметом обговорення в даному розділі.

Після того, як в Паскалі з'явилось поняття "об'єкт", на його основі було створено ряд нових типів, які дозволили абстрагуватись від конкретного поняття "файл". За допомогою об'єктного підходу можна однаково працювати з будь-яким

зовнішнім сховищем даних (дискові накопичувачі, різні види пам'яті і т.і.). Типи файлів засновані на базовому класі TStream (потік), який має набір віртуальних методів запису та зчитування інформації і встановлення конкретної позиції всередині набору даних.

Клас TFileStream, призначений для роботи з файлами на жорсткому диску, є нащадком класу THandleStream. На ньому базуються класи, призначені для зчитування та запису інформації при роботі з об'єктами. Звертатись до цих об'єктів можна через універсальний ідентифікатор Windows - handle. Оскільки при об'єктному підході файл представляється послідовністю (потоком) байт, то він може опрацьовуватись лише як двійковий. Процес роботи з файлом, представленим в програмі у вигляді об'єкту, аналогічний звичайному процесу роботи з файлом. Спочатку файл відкривається (для об'єкта цю функцію виконує конструктор), потім відбувається запис або зчитування даних, на завершення файл закривається (викликається деструктор).

Конструктор має два параметри: ім'я файлу, що відкривається, і режим відкриття:

```
Constructor Create (const FileName:string; Mode:Word);
```

Параметр Mode приймає одне з наступних значень:

1) fmCreate - створюється новий файл або існуючий файл відкривається в режимі запису;

2) fmOpenRead - файл відкривається для читання;

3) fmOpenWrite - файл відкривається для запису;

4) fmOpenReadWrite - файл відкривається для читання і запису.

Для закриття файлу викликається метод Free.

Для зчитування даних з потоку застосовується метод

```
Function Read (var Buffer; Count:Longint): Longint.
```

У змінну Buffer, починаючи з поточної позиції в файлі, записується кількість байт, вказана в параметрі Count. Функція повертає реально зчитану кількість байт.

Запис даних здійснюється за допомогою методу

```
Function Write (const Buffer; Count:Longint): Longint.
```

Кількість байт, вказана в параметрі `Count`, записується зі змінної `Buffer` в поточну позицію в файлі. Функція повертає реально записану кількість байт.

Метод

```
Function Seek (Offset:Longint; Origin:Word): Longint;
```

дає можливість встановити поточну позицію у файлі в залежності від параметру `Offset` (кількість байт, на яку зміщується ця позиція) і параметру `Origin`, який може приймати значення `soFromBeginning` (від початку файлу), `soFromCurrent` (від поточної позиції у файлі), `soFromEnd` (від кінця файлу).

Власних властивостей у класу `TFileStream` немає. Він наслідує дві властивості класу `THandleStream`: ідентифікатор `Handle` і властивість `Size`, яка визначає довжину файлу в байтах. Ще одна властивість успадкована від класу `TStream` - це поточна позиція файлового вказівника, відрахована в байтах від початку файлу (властивість `Positiion`).

Клас `TFileStream` наслідує також ряд методів класів-предків. Від класу `THandleStream` успадковано метод

```
Procedure SetSize (NewSize:Longint);
```

який дозволяє змінити поточний розмір файлу на новий, який, як правило, менший поточного.

Для копіювання файлів зручно застосовувати метод

```
Function CopyFrom (Source:TStream; Count:Longint): Longint;
```

базового класу `TStream`. Перший параметр - потік-джерело, з якого береться кількість байт, задана параметром `Count`, і копіюється в поточний об'єкт. При цьому автоматично виконуються операції по тимчасовій буферизації даних.

3.3.2 Стандартні діалогові вікна Windows

На панелі Dialogs розташовано ряд невізуальних компонентів, які дозволяють використовувати у програмі стандартні діалогові вікна, наприклад, вікна вибору і зберігання файлів або зображень, вікна вибору кольору і шрифта, вікно налагодження принтера, та інші. Ці компоненти не призначені для виконання конкретних дій: завантаження файлу, друку, зміни поточного шрифта та інших. Вони застосовуються лише для одержання від користувача бажаних значень налагоджень, наприклад, введення повного імені файлу разом зі шляхом пошуку, вказання гарнітури шрифта, задавання кількості друкованих сторінок. Всі ці компоненти є нащадками класу TCommonDialog. Найважливіший його метод - це функція

```
function Execute: Boolean;
```

Вона виконує відкриття відповідного вікна і повертає значення true, якщо користувач клікнув на кнопці OK. Поля введення і заголовки визначаються у компонентах. Коли діалогове вікно відкривається вперше, генерується подія OnShow, а при закритті вікна - подія OnClose.

Компонент TOpenDialog (Вікно вибору файлу) призначений для вибору файлу з метою наступного відкриття. Властивості і події класу TOpenDialog [28] наведені в таблицях 3.3.13, 3.3.14.

Таблиця 3.3.13 - Властивості класу TOpenDialog

Властивість	Призначення
(1)	(2)
DefaultExt	Розширення імені файлу, використовуване за замовчуванням
FileName	Обране користувачем ім'я файлу разом з повним шляхом пошуку
Files	Список обраних імен файлів
HistoryList	Список раніше обраних файлів
InitialDir	Поточний каталог, вміст якого відображається при першому відкритті

(1)	(2)
Filter	Набір масок, відповідно до яких відбираються імена файлів для відображення в діалоговому вікні. Кожна маска складається з двох частин - назви і шаблону, які розділяються символом |.
FilterIndex	Номер поточної маски (нумерація починається з 1)
Options	Набір прапорців, які визначають роботу вікна вибору файлів
Title	Заголовок діалогового вікна

Таблиця 3.3.14 - Події класу TOpenDialog

Подія	Умова генерації
OnCanClose	Користувач намагається закрити діалогове вікно. Опрацьовувач цієї події дозволяє проконтролювати правильність обраного імені файлу і дозволити або заборонити закриття
OnFolderChange	Користувач перейшов до іншого каталогу
OnSelectionChange	Користувач обрав нове ім'я файлу в діалоговому вікні
OnTypeChange	Користувач обрав нову маску файлів
OnIncludeItem	До поточного списку файлів в діалоговому вікні буде додано нове ім'я. Опрацьовувач даної події дає можливість відбирати допустимі імена за алгоритмом, який визначає програміст

Серед методів цього класу слід відзначити функцію

```
Function GetStaticRect: TRect;
```

Вона повертає координати прямокутної області діалогового вікна (частина клієнтської області), зарезервованої для потреб розробника (наприклад, для відображення вмісту поточного обраного файлу).

Компонент `TSaveDialog` (Вікно збереження файлу) практично нічим не відрізняється від компонента `TOpenDialog` за виключенням деяких налагоджень, специфічних для процесу збереження файлу.

Компоненти `TOpenPictureDialog` (Вікно відкриття рисунка) і `TSavePictureDialog` (Вікно збереження рисунка) є, відповідно, нащадками класу `TOpenDialog` і класу `TSaveDialog`. Діалогові вікна містять додаткову область для швидкого перегляду вмісту обраного графічного файлу.

Компонент `TFontDialog` (Вікно вибору шрифта) призначений для виклику стандартного діалогового вікна вибору шрифта, доступного в системі. Відповідно до полів цього вікна компонент має набір властивостей, наведених в таблиці 3.3.15.

Таблиця 3.3.15 - Властивості компонента `TFontDialog`

Властивість	**Призначення**
`Device`	Пристрій, для якого відображається список доступних шрифтів
`MaxFontSize`	Максимальний розмір шрифта, який обмежує вміст відображуваного списку шрифтів
`MinFontSize`	Мінімальний розмір шрифта, який обмежує вміст відображуваного списку шрифтів
`Font`	Обраний користувачем шрифт
`Options`	Додаткові характеристики зовнішнього вигляду діалогового вікна

Компонент `TColorDialog` (Вікно вибору кольору) викликає стандартне діалогове вікно вибору кольору (рис.3.3.1). Властивість `Color` містить обраний користувачем колір, а властивість `CustomColors` зберігає в текстовому форматі опис додаткових кольорів. Колір в цьому форматі задається шістьма символами, які визначають в шістнадцятковому вигляді значення кольору відповідно до вимог системи RGB. Кожен байт задається двома символами. Крім того, є властивість `Options`, яка дозволяє виконувати специфічні налагодження вікна.

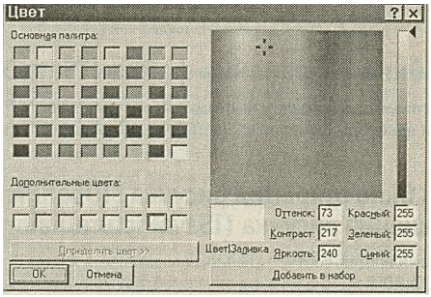

Рис.3.3.1 - Стандартне діалогове вікно вибору кольору в операційній системі Windows

Компонент `TPrinterSetupDialog` (Налагодження принтера) призначений для налагодження параметрів роботи принтера, не має оригінальних властивостей, тому що ці налагодження істотно відрізняються для різних видів принтерів. На основі цього компонента можна створювати власні компоненти для окремих принтерів.

Компонент `TPrintDialog` (Друк) відображає стандартне вікно друку Windows. У ньому можна задати різні параметри друку, які визначаються властивостями, наведеними в таблиці 3.3.16.

Таблиця 3.3.16 - Властивості компонента `TPrintDialog`

Властивість	**Призначення**
`Collate`	Прапорець "Розібрати за копіями"
`Copies`	Кількість копій
`FromPage`	Номер сторінки, з якої починається друк
`MaxPage`	Максимальна кількість сторінок, які можуть бути надруковані
`MinPage`	Мінімальна кількість сторінок, які можуть бути надруковані
`Options`	Додаткові параметри налагодження
`PrintRange`	Діапазон друкованих сторінок документу
`PrintToFile`	Має значення true, якщо виведення повинно здійснюватись не на принтер, а в файл
`ToPage`	Номер сторінки, на якій завершується друк

Компонент `TFindDialog` (Пошук) використовується для відображення діалогового вікна пошуку текстового рядка. Властивості класу `TFindDialog` наведені в таблиці 3.3.17.

Таблиця 3.3.17 - Властивості класу `TFindDialog`

Властивість	**Призначення**
`FindText`	Рядок для пошуку
`Options`	Додаткові налагодження
`Position`	Координата лівого верхнього кута діалогового вікна при його виведенні на екран (в пікселях)

З методів цього класу слід відзначити процедуру

```
Procedure CloseDialog;
```

Ця процедура закриває вікно, але не змінює значень встановлених властивостей, щоб надалі можна було виконати повторний пошук з такими ж параметрами.

Компонент `TReplaceDialog` (Пошук і заміна) є нащадком класу `TFindDialog`. Він дещо розширює його можливості і дозволяє вводити рядок для заміни знайденого тексту. Компонент має нову властивість `ReplaceText` і відповідне їй поле в діалоговому вікні.

Друк з програми. Виведення довільної інформації на друк реалізовано в Delphi за допомогою об'єкта `Printer` класу `TPrinter` (модуль `Printers`), який містить властивість `Canvas` типу `TCanvas` ("холст"). При відображенні на ньому довільної інформації вона виводитиметься не на екран, а на принтер. Властивість `Canvas` об'єкту `Printer` доступна, лише коли принтер готовий до друку даних. Для підготовки властивості `Canvas` використовуються методи

```
Procedure BeginDoc;
Procedure EndDoc;
```

Перший з них визначає початок друку і створює екземпляр класу `TCanvas`, а другий вказує на завершення друку, після чого властивість `Canvas` буде недоступна.

При виведенні графічних даних на друк слід враховувати, що розрішення екрану (в пікселях) і принтера (в точках) істотно розрізняються.

Тому потрібно створити окрему процедуру, яка виконуватиме виведення графічних даних на область принтера або форми з врахуванням коефіцієнтів масштабування на основі поточного розрішення екрана та принтера.

Для виведення лише текстової інформації можна використовувати інший підхід. Стандартна процедура `AssignPrn` зв'язує файлову змінну типу `TextFile` з поточним принтером. При цьому процедури `Write` і `Writeln` виводитимуть рядок тексту одразу на принтер, відповідно, починаючи друк з поточної позиції рядка або з нового рядка.

Властивості і методи класу `TPrinter` наведені в таблицях 3.3.18, 3.3.19.

Таблиця 3.3.18 - Властивості класу `TPrinter`

Властивість	**Призначення**
`Aborted`	Має значення true, якщо користувач перервав процес друку
`Capabilities`	Налагодження режиму друку
`Canvas`	Область виведення графічної інформації для принтера
`Copies`	Кількість друкованих копій
`Fonts`	Список шрифтів, які підтримуються поточним принтером
`Orientation`	Орієнтація паперу: книжкова чи альбомна
`PageHeight`	Висота друкованої сторінки в пікселях
`PageNumber`	Номер поточної друкованої сторінки
`PageWidth`	Ширина друкованої сторінки в пікселях
`PrinterIndex`	Номер принтера з властивості `Printers`
`Printers`	Список назв всіх принтерів, доступних системі
`Printing`	Має значення true, коли виконується друк
`Title`	Стандартний заголовок сторінки

Таблиця 3.3.19 - Методи класу `TPrinter`

Метод	Призначення
`Procedure Abort;`	Переривання друку
`Procedure GetPrinter (ADevice, ADriver, APort:PChar; var ADeviceMode:THandle);`	Одержання інформації про поточний принтер
`Procedure NewPage;`	Початок друку нової сторінки
`Procedure Refresh;`	Оновлення списку шрифтів та принтерів, встановлених в системі
`Procedure SetPrinter (ADevice, ADriver, APort:PChar; var ADeviceMode:THandle);`	Вказаний принтер стає поточним

3.3.3 Панелі `Additional`, `Win32`, `System` середовища Delphi

Додаткові компоненти Delphi (*панель* `Additional`):

1) Компонент `TSpeedButton` (Швидка кнопка) використовується при формуванні панелей управління з "швидкими" комнадними кнопками.

2) Компонент `TBitBtn` (Кнопка з картинкою) призначений для створення кнопки з картинкою. В системі є набір готових шаблонів.

3) Компонент `TMaskEdit` (Шаблон введення) дозволяє вводити дані в текстове поле за заданим шаблоном. Він корисний для додатків, де потрібно контролювати введену користувачами інформацію.

4) Компонент `TBevel` (Рамка) використовується для створення рамок і окремих ліній оформлення. Нагадує панель, але не призначений для групування елементів.

5) Компонент `TStaticText` (Постійний текст) - основна відмінність цього компонента від `TLabel` лише в тому, що він дозволяє брати текст, що виводиться, в рамку.

6) Компонент `TShape` (Фігура) призначений для відображення на формі різних геометричних фігур (рис.3.3.2).

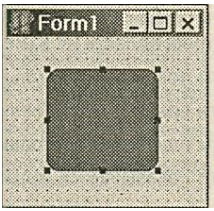

Рис.3.3.2 - Фігура (квадрат із заокругленими кутами), розташована на формі

7) Компонент `TSplitter` (Роздільник) - за його допомогою клієнтська область форми може бути розділена на декілька панелей, розміри яких допускається змінювати, перетягуючи границі цих панелей (рис.3.3.3).

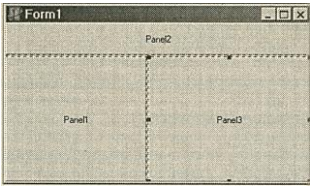

Рис.3.3.3 - Форма, яка містить панелі, зв'язані роздільниками

8) Компонент `TApplicationEvents` (Події додатку) може використовуватись сумісно з компонентом `TActionList`. Він дозволяє приймати та опрацьовувати всі повідомлення, адресовані додатку.

9) Компонент `TStringGrid` (Таблиця рядків) дозволяє працювати з текстовою інформацією в двовимірній таблиці, яка має стовпці і рядки, розміри яких можна змінювати за допомогою миші. До кожної комірки таблиці можна "прив'язати" свій об'єкт, характеристики якого представляються у вигляді рядка, розташованого в цій комірці (рис.3.3.4).

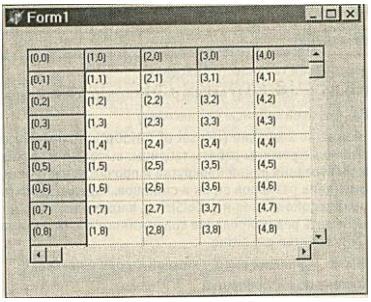

Рис.3.3.4 - Форма, яка містить заповнену таблицю рядків

10) Компонент `TDrawGrid` (Рисована таблиця) дозволяє зберігати в комірках таблиці довільні об'єкти. Вся робота по візуальному представленню кожного об'єкта покладається на програміста.

11) Компонент `TCheckListBox` (Список з прапорцями) - список з прапорцями нічим не відрізняється від звичайного списку за виключенням додаткових прапорців на початку кожного рядка.

12) Компонент `TScrollBox` (Область прокрутки) дозволяє організувати в рамках однієї форми необмежену кількість областей прокрутки з оригінальним вмістом (рис.3.3.5).

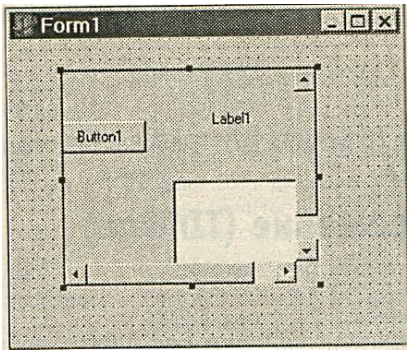

Рис.3.3.5 - Прокручувана область з елементами управління

13) Компонент `TImage` (Зображення) використовується в багатьох програмах, причому не лише для відображення статичних картинок, але й для створення анміаційних ефектів.

14) Компонент `TChart` (Діаграма) дозволяє будувати двота тривимірні діаграми на основі різних даних (рис.3.3.6, 3.3.7).

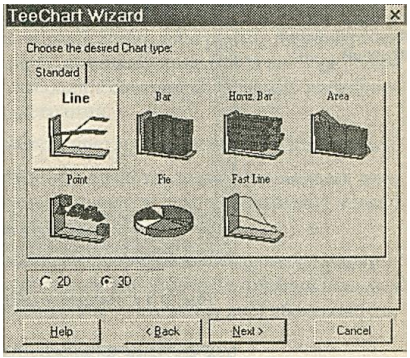

Рис.3.3.6 - Види тривимірних діаграм

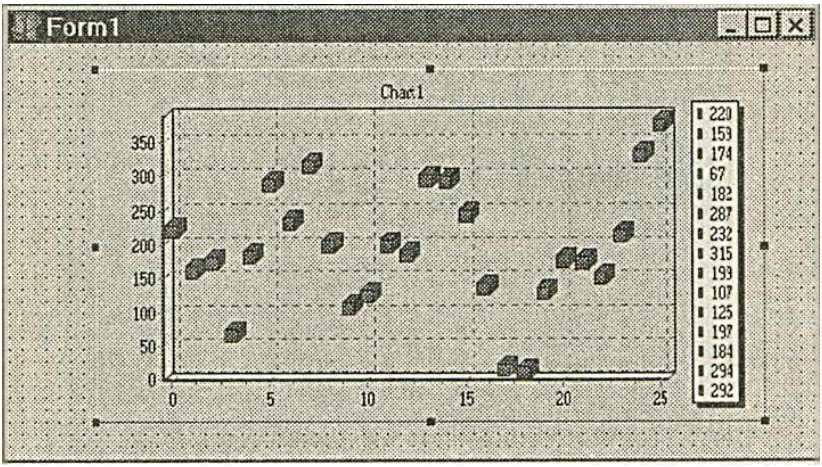

Рис.3.3.7 - Форма з автоматично згенерованим об'єктом-діаграмою

Класи та компоненти панелі `Win32`:

1) Клас `TList` (Список) представляє собою масив вказівників на об'єкти довільних типів. За його допомогою можна додавати до списку нові об'єкти, видаляти їх, знаходити, сортувати та перевпорядковувати.

2) Клас `TCollection` (Колекція) призначений для роботи з елементами однакової структури - вказівниками

конкретного типу, в результаті чого підвищується ефективність опрацьовування збережуваної інформації.

3) Компонент `TPageControl` (Набір сторінок) представляє собою набір сторінок, накладених одна на одну. Доступ до кожної сторінки, яка містить свій набір елементів управління, здійснюється через вкладки - невеличкі виступи над сторінкою, які містять коротку назву. Більшість налагоджувальних діалогових вікон в різних програмах для Windows створені саме за таким принципом (вікно Параметри редактора Word, вікно Властивості браузера Internet Explorer, вікно Налагодження браузера Mozilla Firefox). Даний елемент управління дозволяє економити екранний простір, фактично необмежено збільшуючи його "глибину" (рис.3.3.8). Усі об'єкти, розташовані на сторінках компоненту `TPageControl`, вважаються такими, що належать безпосередньо формі-предку - класу `TForm1`, тому явно вказувати сторінки при звертанні до цих об'єктів не обов'язково.

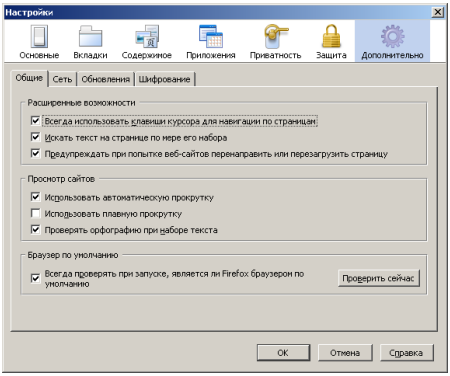

Рис.3.3.8 - Сторінки вкладок вікна Налагодження браузера Mozilla Firefox

4) Компонент `TTabControl` (Сторінки, що перемикаються) має спільні риси з набором сторінок. Головна відмінність полягає в тому, що, при наявності декількох вкладок, реально доступна лише одна сторінка і лише одна клієнтська область. Тобто, користувач може перемикати лише вкладки, але не сторінки. При перемиканні вкладок вміст сторінки не змінюється автоматично. Даний компонент використовується, коли потрібно

реалізувати складну логіку роботи такого елемента управління. Контролювати вміст кожної "віртуальної" сторінки потрібно програмно, тобто динамічно створювати і знищувати або робити невидимими різні групи елементів та графічні зображення в моменти перемикання між вкладками.

5) Компонент `TImageList` (Список зображень) використовується при створенні додатків для роботи з графічними зображеннями. Він дозволяє зберігати набори картинок фіксованого розміру, звертатись до них за номерами та здійснювати виведення зображень на екран різними способами.

6) Компонент `TRichEdit` (Текстовий редактор) представляє собою стандартний елемент управління. В порівнянні з компонентом `TMemo` має такі розширені можливості, як форматування окремих абзаців тексту, підтримка формату RTF та інші.

7) Компонент `TTrackBar` (Повзунок) застосовується там, де потрібно в візуальному режимі виставити за допомогою миші наближене значення, що виконується перетягуванням повзунка по шкалі (рис.3.3.9).

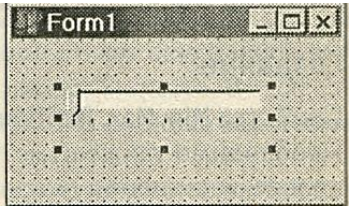

Рис.3.3.9 - Повзунок в процесі формування форми

8) Компонент `TProgressBar` (Індикатор) застосовується для відображення відомостей про хід процесів (наприклад, процесу інсталяції).

9) Компонент `TUpDown` (Лічильник) використовується у комплексі з іншими елементами. Дає можливість змінювати числові величини за допомогою кнопок зі стрілками та клавіш курсору. Можна задати допустимі межі цих величин.

10) Компонент `THotKey` (Гаряча клавіша) дозволяє запитати в користувача певну "гарячу" комбінацію клавіш, яка надалі буде використана для виклику часто виконуваної дії.

11) Компонент `TAnimate` (Анімація AVI) дозволяє організовувати на формі невелику анімацію - відтворити кліп в форматі AVI без відтворення звуку (рис.3.3.10).

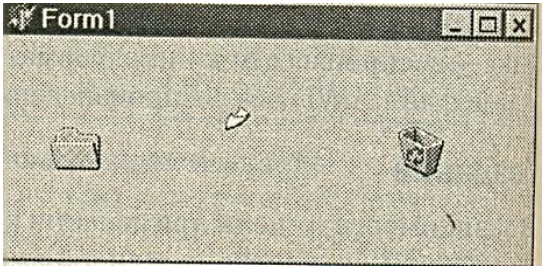

Рис.3.3.10 - Приклад анімаційного кліпу без звуку

12) Компонент `TMonthCalendar` (Календар) дозволяє вибрати потрібну дату за допомогою миші (рис.3.3.11).

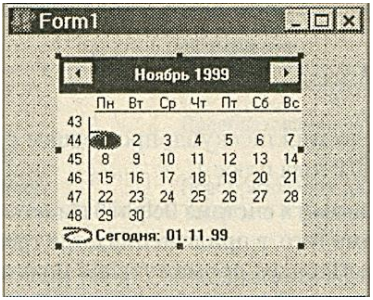

Рис.3.3.11 - Календар, розташований на формі

13) Компонент `TDateTimePicker` (Поле введення дати/часу) представляє собою список, що розкривається, і використовується для введення дати і часу з клавіатури (за форматом відповідно до локальних налагоджень Windows).

14) Компонент `THeaderControl` (Панель заголовків) дозволяє розташувати на формі заголовки довільних елементів. Порядок і розміри цих заголовків можна змінювати,

підлаштовувати розміри інших об'єктів під розміри розділів заголовку.

15) Компонент `TStatusBar` (Рядок стану) містить "гарячу" підказку і виводить додаткову інформацію. Рядок стану звичайно поділяється на декілька панелей (рис.3.3.12).

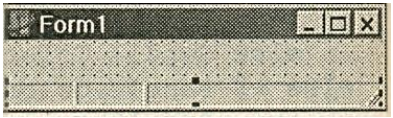

Рис.3.3.12 - Рядок стану в процесі формування форми

16) Компонент `TToolBar` (Панель інструментів) використовується для управління швидкими командними кнопками та іншими "гарячими" елементами управління. Дозволяє формувати багаторядкові набори інструментів.

17) Компонент `TControlBar` (Панель управління) - розташовано на панелі `Additional`. Він спеціально перенесений в цей розділ, оскільки використовується звичайно разом з об'єктами типу `TToolBar`. При розміщенні такого об'єкту на панелі управління, до нього додається спеціальний корінець, смуга перетягування, розташована ліворуч. За допомогою корінця цей компонент можна перетягувати в межах панелі управління.

18) Компонент `TCoolBar` (Розширена пам'ять управління) об'єднує в собі можливості компонентів `TToolBar` і `TControlBar`. Представляє собою колекцію "плаваючих" панелей, вміст та зовнішній вигляд яких формуються програмістом.

19) Компонент `TPageScroller` (Прокрутка сторінок) дозволяє задавати видиму область для різних елементів управління. Доступ до невидимої частини здійснюється за допомогою стрілок, які автоматично з'являються по краях області прокрутки. Даний компонент відрізняється від компонента `TScrollBox` наявністю кнопок зі стрілками та можливістю задавати напрямок прокрутки лише в горизонтальному або лише у вертикальному напрямку.

20) Компонент `TListView` (Список елементів) містить значно більше можливостей представлення інформації, ніж стандартний список рядків `TListBox`. Компонент орієнтований на представлення даних у вигляді структури "об'єкт - набір властивостей", наприклад, файлів разом з розміром, датою створення, атрибутами. Використовувати його як список однорідної інформації некоректно. Форма з таким списокм елементів представлена на рис.3.3.13.

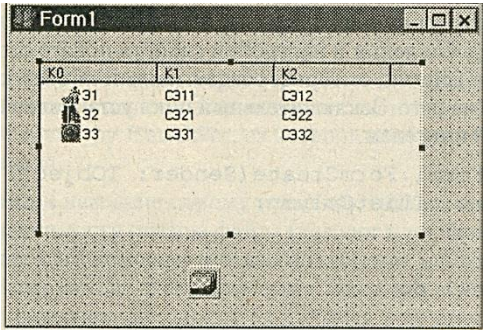

Рис.3.3.13 - Форма, яка містить підготовлений список елементів

21) Компонент `TTreeView` (Дерево) призначений для створення дерев, які відображають ієрархічні структури даних.

Системні компоненти (панель `System`):

1) Компонент `TTimer` (Таймер) вмикає генерацію повідомлень, подібних системному повідомленню від системного таймера `WM_TIMER`, з заданою періодичністю (в мілісекундах) і дозволяє виконувати певну частину обчислень саме в опрацьовувачі цієї події. За допомогою цього компонента система може контролювати ресурси в момент одержання програмою системного повідомлення, що особливо важливо для програм, які виконують дії, пов'язані з моделюванням або опрацюванням графіки, призначених для спілкування з користувачем в реальному часі або тривалих обчислень.

2) Компонент `TMediaPlayer` (Мультимедійний програвач) містить ряд можливостей Універсального програвача та призначений для відтворення в рамках програми музичних та

відеокліпів в різних форматах, які підтримуються драйвером MCI. Управління відтворенням здійснюється за допомогою набору кнопок, які нагадують кнопки музичних центрів (рис.3.3.14).

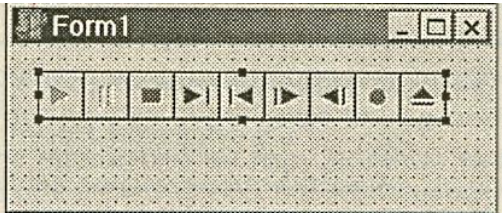

Рис.3.3.14 - Розташування мультимедійного програвача на формі

Вправи з розв'язками

Вправа 1. Написати програму мовою С++, яка дозволяє розглянути виведення у вікно з використанням об'єктно-орієнтованого інтерфейсу користувача GDI+ та відображає 2 прямокутника: один при опрацюванні повідомлення WM_PAINT, інший - при опрацюванні повідомлення WM_LBUTTONDOWN.

Текст програми:

```
#include "Functions.h"
#define UNICODE
#include <gdiplus.h>
using namespace Gdiplus;
#pragma comment (lib, "gdiplus.lib")
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM,
LPARAM);
HINSTANCE hInstance;
char szClass[]="MyGDIPlus";
VOID Example(HDC hdc)
{Graphics graphics(hdc);
Pen pen(COLOR(255, 0, 0, 255), 2);
graphics.DrawRectangle(&pen, 5, 5, 50, 100); }
VOID Example1 (HDC hdc, int l, int t)
{Graphics graphics(hdc);
Pen pen(Color(255, 255, 0, 0), 4);
graphics.DrawRectangle(&pen, l, t, 70, 80); }
INT WINAPI WinMain(HINSTANCE hInstance,
HINSTANCE, PSTR, INT nCmdShow)
{ MSG msg; HWND hwnd; ::hInstance=hInstance;
GdiplusStartupInput gdiplusStartupInput;
ULONG_PTR gdiplusToken;
GdiplusStartup(&gdiplusToken,
&gdiplusStartupInput, NULL);
if (!RegClass(WndProc, szClass, COLOR_WINDOW))
return FALSE;
hwnd=CreateWindow (szClass, "MyGDIPlus",
WS_OVERLAPPEDWINDOW|WS_VISIBLE,
CW_USEDEFAULT,
```

```
CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
0, 0, hInstance, NULL);
if (!hwnd) return FALSE;
while    (GetMessage(&msg,    NULL,    0,    0))
DispatchMessage(&msg);
GdiplusShutdown(gdiplusToken);
return(int)msg.wParam; }
LRESULT CALLBACK WndProc(HWND hwnd, UINT msg,
WPARAM wParam, LPARAM lParam)
{switch(msg)
// опрацювання повідомлення WM_PAINT
{case WM_PAINT:
{PAINTSTRUCT ps; HDC hdc=BeginPaint(hwnd, &ps);
Example(hdc);
EndPaint(hwnd, &ps); return 0; }
// опрацювання повідомлення WM_LBUTTONDOWN
case WM_LBUTTONDOWN:
{int l, t; HDC hdc=GetDC(hwnd);
l=LOWORD(lParam); t=HIWORD(lParam);
Example1(hdc, l, t);
ReleaseDC(hwnd, hdc); return 0; }
case WM_DESTROY: {PostQuitMessage(0); return 0;}
}   return   DefWindowProc(hwnd,   msg,   wParam,
lParam);}
```

Вправа 2. Охарактеризуйте декілька об'єктів та класів об'єктно-орієнтованого інтерфейсу користувача GDI+, які призначені для малювання геометричних фігур мовою C++.

Об'єкт `Color`. Об'єкт `Color` зберігає 32-розрядне значення кольору. Колір задають за допомогою чотирьох 8-розрядних компонент: `alpha`, `red`, `green`, `blue`. Компоненти приймають значення від 0 до 255. Компоненти `red`, `green`, `blue` описують інтенсивність відповідно червоного, зеленого і синього кольорів. Параметр `alpha` задає ступнь прозорості кольору (0 - повністю прозорий колір, 255 - непрозорий колір). Екземпляр об'єкту `Color` створю один з наступних

конструкторів: `Color(), Color(a,r,g,b), Color(argb), Color(r,g,b).`

Об'єкт `Pen`. Об'єкт `Pen` (перо) використовують для малювання ліній і прямокутників. Він має два конструктори - `Pen(brush,width)` і `Pen(color,width)`. Параметр `width` за замовчуванням дорівнює 1.0. Перший конструктор створює об'єкт `Pen`, який для малювання ліній використовує пензль `brush` і товщину ліній `Width`. Другий конструктор створює об'єкт `Pen` кольору `color` і товщини ліній `width`.

*Об'єкт `Brush`. А*бстрактний клас, що є основою більш досконалих пензлів. Він має 3 методи.

Клас `SolidBrush`. Визначає об'єкт `Brush` одного суцільного кольору. Він має єдиний конструктор `SolidBrush(const Color& color)`, який створює пензель суцільного кольору. Наслідує методи класу `Brush` і має 2 власних методи.

Клас `HatchBrush`. Визначає штрихований об'єкт `Brush`. Має єдиний конструктор: `HatchBrush(HatchStyle hatchStyle, const Color& foreColor, const Color& backColor)`, який створює пензель з кольором фону `backColor`, кольором ліній `foreColor` та стилем ліній `hatchStyle`. Він наслідує методи класу `Brush` і має 3 власних методи.

Клас `LinearGradientBrush`. Цей клас визначає пензель зі змінним кольором, де колір поступово змінюється від початкової граничної лінії до кінцевої граничної лінії. Граничні лінії паралельні одна одній, а градієнт кольору перпендикулярний їм. Він має 6 конструкторів. Різниця між ними полягає в способі задання початкової і кінцевої точок градієнту. Найпростіший з них: `LinearGradientBrush(const Point& point1, const Point& point2, const Color& color1, const Color& color2)` створює об'єкт з вказанням почтакових та кінцевих точок і кольорів градієнту кольору. Він наслідує методи класу `Brush` і має 15 власних методів.

Клас `Image`. Клас `Image` містить методи завантаження і зберігання растрових і векторних зображень. Об'єкт `Image` можна

створювати з різного типу файлів: BMP, ICON, GIF, JPEG, Exif, PNG, TIFF, WMF, EMF. Він має 2 конструктори, які відрізняються способом завантаження зображення. Конструктор завантаження з файлу: `Image(const WCHAR* filename, BOOL useEmbeddedColorManagement);` створює об'єкт `Image` з зображення, яке зберігається в файлі filename. Має 37 методів.

Вправа 3. Створити засобами мови C++ Builder додаток, який містить дві форми - на першій формі розташована кнопка, при натисканні на яку відображається друга форма.

1. Створимо новий проект `File | New Application`.
2. Змінимо значення властивості `Name`, наприклад, на `MainForm`, а властивості `Caption` - на `Multiple Forms Test Program`.
3. Збережемо проект. Для модуля задамо ім'я `Main`, а для проекту - `Multiple`.
4. Розташуємо кнопку на формі. Властивості `Name` надамо значення `ShowForm2`, а властивості `Caption` - значення `Show Form 2`.
5. Створимо нову форму за допомогою пункту `File | New Form` головного меню або кнопки `New Form` панелі інструментів. Після створення форма матиме ім'я `Form1` і розташується поверх головної форми.
6. Змінимо розмір і положення нової форми так, щоб вона була вдвічі меншою головної форми і розташовувалась в її центрі. Для переміщення форми використаємо рядок заголовкка. Розмір змінимо пересуванням нижнього правого кута.
7. Змінимо значення властивості `Name` на `SecondForm`, а значення `Caption` - на `A Second Form`.
8. Збережемо файл під іменем `Second`.
9. Оберемо компонент `Label` і розташуємо його на `SecondForm`. Змінимо текст властивості `Caption` на `This is`

the second form. Відцентруємо це повідомлення відносно форми.

10. Клікнемо на головній формі. Друга форма тепер закрита головною формою. Двічі клікнемо на кнопці Show Form 2. На екрані з'явиться вікно редактора коду, і курсор буде розташований там, де потрібно починати введення коду для опрацювання цієї кнопки.

11. Введемо наступний код:

```
void _fastcall TMainForm::ShowForm2Click
(TObject *Sender)
{SecondForm->ShowModal();}
```

12. Якщо зараз запустити програму, то одержимо повідомлення про помилку. Це відбувається через те, що в модулі MainForm немає оголошення змінної SecondForm, яка є вказівником на екземпляр класу TSecondForm. Потрібно вказати, де розташовано оголошення цього класу, для чого слід включити заголовочний файл (header file) для SecondForm у вихідний файл MainForm за допомогою директиви #include. Оберемо в головному меню пункт File | Include Unit Hdr. На екрані з'явиться діалогове вікно Include Unit, в якому знаходиться список доступних модулів. Клікнемо на імені Second, потім на кнопці ОК. Тепер в модуль Main включено оголошення класу з модуля Second. Результат виконання програми показано на рис.3.1.

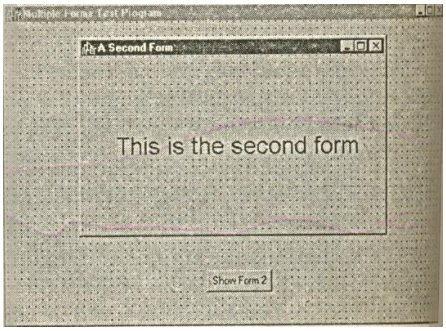

Рис.3.1 - Вікно програми, яка відображає другу форму при натисканні на кнопку першої форми

Вправа 4. Створити мовою C++ Builder демонстраційний прототип додатку, схожого на Windows Notepad (Блокнот).

1. Створимо новий додаток - в головному меню File оберемо пункт New Application, змінимо значення властивості Name на ScratchPad, а значення властивості Caption - на Scratch Pad 1.0. В головному меню оберемо пункт Project | Options і на вкладці Application введемо ScratchPad 1.0 в якості заголовка додатку.

2. Створимо роздільник панелі інструментів в верхній частині вікна - оберемо компонент Bevel (вкладка Additional) з палітри компонентів і помістимо його на форму, встановимо для властивості Height значення 2, змінимо значення Align на alTop. Лінія розташується вздовж верхнього краю робочої області форми.

3. Створимо контейнер для кнопок панелі інструментів - оберемо компонент Panel з палітри компонентів і розмістимо його в будь-якому місці форми, змінимо значення властивості Name на Toolbar, встановимо значення властивості Height рівним 32, змінимо значення властивості BevelOuter на bvNone, видалимо значення властивості Caption, змінимо значення властивості Align на alTop. Панель переміститься вгору вікна.

4. Заповнимо панель - додамо кнопку FileOpen: клікнемо на вкладці Additional палітри компонентів і оберемо компонент SpeedButton; розташуємо кнопку на панель (не на форму!); змінимо значення властивості Name на FileOpenBtn; встановимо значення властивості Left рівним 5; оберемо в головному меню пункт View | Alignmnet Palette і клікнемо на кнопці Center Vertically in Window; для властивості Glyph двічі клікнемо в стовпці Value, після чого відкриється вікно редактора зображень; натиснемо кнопку Load і оберемо файл fileopen.bmp (\Program Files \ Common Files \ Borland Shared \ Images \ Buttons). Виконаємо ці ж кроки для додавання кнопки File Save праворуч від кнопки File Open (властивість Name – FileSaveBtn).

5. Додамо рядок стану - клікнемо на вкладці Win32 палітри компонентів і оберемо компонент StatusBar, клікнемо в будь-якому місці форми (рядок стану автоматично розташується в нижній частині форми), змінимо значення властивості Name на StatusBar.

6. Додамо компонент Memo, в якому можна набирати текст - помістимо на форму компонент Memo з вкладки Standard палітри компонентів; змінимо значення властивості Name на Memo; двічі клікнемо в стовпці Value поруч із властивістю Lines, після чого відкриється вікно редактора рядків, в якому слід витерти слово Memo; змінимо значення властивості ScrollBar на ssVertical; встановимо системний шрифт Fixedsys (атрибут Name властивості Font); змінимо значення Align на aClient для заповнення компонентом всієї робочої області форми між пенллю інструментів та рядком стану

7. Запустимо програму - одержимо вікно, зображене на рис.3.2. В робочій області вікна можна набирати текст. Збережемо проект File | Save All.

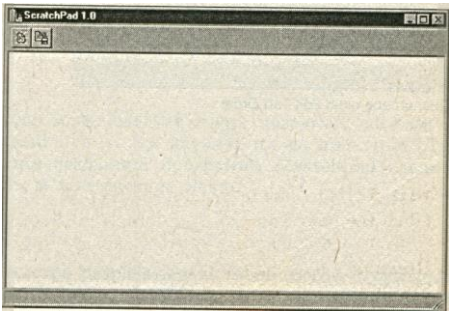

Рис.3.2 - Прототип програми Блокнот

Додамо до створеного прототипу меню:

1. Розташуємо на форму компонент MainMenu і змінимо значення його властивості Name на MainMenu, двічі клікнемо на значку головного меню, і на екрані з'явиться редактор меню. Змінимо значення властивості Name на FileMenu.

2. Клікнемо на властивості Caption у вікні інспектора об'єктів, наберемо &File і натиснемо Enter. Після цього в редакторі меню з'явиться пункт File, під яким і праворуч від якого з'явились нові порожні пункти.

3. Змінимо значення властивості Name нижнього нового пункту на File New, а значення властивості Caption на &New. Редактор меню знову створить порожній пункт. Повторимо цей крок для створення пунктів меню Open, Save, SaveAs.

4. Створимо роздільник меню - встановимо для властивості Caption нижнього порожнього пункуту меню значення "-" (дефіс).

5. Додамо ще два пункти меню - Print, Print Setup (крок 3), потім - роздільник меню (крок 4), і ще один пункт - Exit (крок 3). На цьому формування пункту меню File завершено.

6. Створимо меню Edit шляхом вставлення його із шаблону - клікнемо на порожньому пункті праворуч від пункту File, потім викличемо контекстне меню і оберемо пункт Insert From Template. З'явиться діалогове вікно Insert Template (рис.3.2.9), в якому приведено список доступних шаблонів. Оберемо зі списку Edit Menu, після чого в редактор одразу ж буде вставлено повне меню Edit, з якого можна видалити непотрібні пункти - наприклад, Repeat<command>, Paste Special (див. 3.2.3), можна додати нові пункти - наприклад, пункт Select All (див. 3.2.3), можна переміщувати пункти - наприклад, пункт SelectAll під пунктом Undo (див. 3.2.3).

7. Вставимо з шаблону меню Help в порожній пункт меню праворуч від Edit - крок 6, зі списку обираємо Help Menu. Результат виконання програми наведено на рис.3.3.

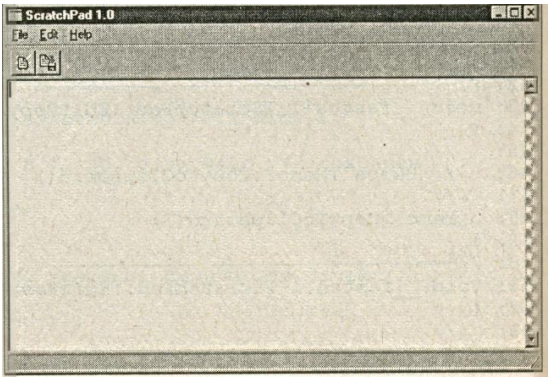

Рис.3.3 - Програма ScratchPad в дії

Тепер у нас є всі пункти меню, але немає коду, який змусив би їх працювати. Більша частина коду вже міститься в класі `TMemo`. Потрібно лише викликати підходящі методи цього класу з опрацьовувачів пунктів меню. До того, як почати писати код, додамо на форму компоненти `OpenDialog` (значення `Name` - `OpenDialog`) та `SaveDialog` (значення `Name` - `SaveDialog`) і розташуємо їх значки на одній лінії зі значком `MainMenu`.

Створимо код, наприклад, для пункту меню `File | Exit`. Редактор меню повинен бути закритий. Оберемо в головному меню форми пункт `File | Exit`. Вікно редактора коду переміститься на передній план, і в ньому буде відображатись опрацьовувач події `FileExitClick()`. З місця, де розташований курсор, введемо:

```
Close();
```

Напишемо код ще для одного пункту меню `Edit | Cut`. В обробнику події `EditCutClick()` введемо:

```
Memo->CutToClipboard();
```

У результаті програма матиме вигляд, показаний на рис.3.3, але при виборі пункту меню `File | Exit` програма закриватиметься, а при виборі пункту меню `Edit | Cut` буде здійснюватись видалення виділеного фрагменту із занесенням його у буфер обміну.

Вправа 5. Виконати програмне відображення таблиці рядків з використанням білих символів на чорному фоні (мова програмування Delphi 7).

Для відображення кожної комірки слід визначити опрацьовувач події `OnDrawCell`.

```
Procedure TForm1.StringGrid1DrawCell
(Sender: Tobject;
ACol, ARow: Integer;
Rect: TRect; State: TGridDrawState);
```

Параметри `ACol, ARow` містять номери стовпця і рядка комірки, що відображається, параметр `Rect` - клієнтські координати області таблиці, яка повинна бути відображена, параметр `State` визначає статус комірки: `gdSelected` (виділена), `gdFocused` (має фокус введення), `gdFixed` (лежить в області заголовку таблиці).

Задати вміст таблиці тепер можна за допомогою наступної процедури (рис.3.4):

```
Procedure TForm1.StringGridDrawCell
(Aender: Tobject;
ACol, ARow: Integer;
Rect: TRect; State: TGridDrawState);
Begin
With Sender as TStringGrid do
Begin
Canvas.Font.Color:=clWhite;
Canvas.Brush.Color:=clBlack;
Canvas.FillRect(Rect):
Canvas.TextOut(Rect.Left+5,Rect.Top+5,
'(' + IntToStr(ACol) + ','
+ IntToStr(ARow) + ')' );
end; end;
```

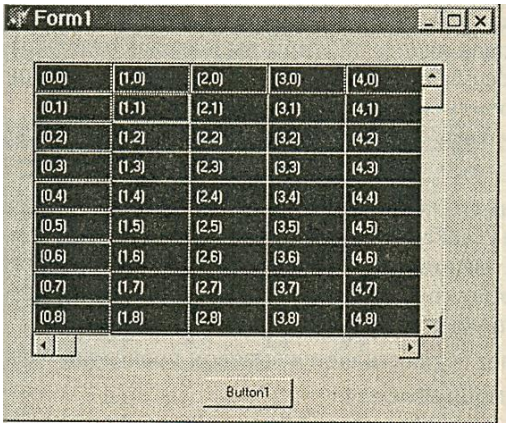

Рис. 3.4 - Відображення таблиці з використанням білих символів на чорному фоні

Вправа 6. Розташувати на формі три компоненти - текстове поле, панель і кнопку та організувати можливість перетягування тексту з текстового поля на панель та переміщення кнопки по формі (мова програмування Delphi 7).

Оберемо компонент `Edit` панелі `Standard` та клікнемо на тому місці форми, де потрібно розташувати текстове поле. Значення властивості `Text` даного об'єкту встановимо в `Edit1`. Аналогічно на формі розташуємо кноку (компонент `Button`). Властивість `Caption` кнопки задамо як `Button1`. Аналогічно розташуємо панель (компонент `Panel`). Властивість `Caption` панелі задамо як `Panel1`. Після цього одержимо форму, зображену на рис.3.5.

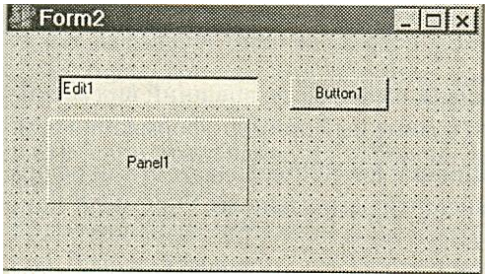

Рис.3.5 - Форма з підготовленими елементами управління

Програмування процесу перетягування здійснюється в декілька етапів:

1) фіксується початок роботи даного процесу за допомогою методу форми `BeginDrag` (наприклад, при натисканні лівої кнопки миші над відповідним об'єктом). Вказівник миші приймає вигляд, вказаний у властивості `DragCursor` (значення за замовчуванням - `crDrag`);

2) відбувається опрацювання процесу перетягування. Для об'єктів, які можуть прийняти перетягуваний об'єкт, задається опрацьовувач події `OnDragOver`. Він викликається, коли вказівник миші в режимі перетягування переміщується над приймаючим об'єктом. Об'єкт повинен відповісти, чи готовий він стати "приймачем", заносячи значення true (якщо готовий) в параметр опрацьовувача `Accept`, який передається за посиланням. В деяких випадках від прийому слід відмовитись, наприклад, якщо в рамках однієї форми підтримується механізм перетягування різних об'єктів і потрібно виконати додаткову перевірку на відповідність типів;

3) коли об'єкт, що перетягується, залишено (кнопку миші відпущено), генерується подія `OnDragDrop`. В ній виконуються всі необхідні дії по опрацюванню інформації, яка зберігається в об'єкті, що перетягується;

4) по завершенню перетягування генерується подія `OnEndDrag`. Її є зміст опрацьовувати лише у випадку перевірки, успішно чи ні виконано перетягування. Для цього параметр `Target`, який зберігає посилання на об'єкт, що приймає, потрібно порівняти зі значенням `nil`. Якщо приймач відсутній, значить, перетягування не виконано.

Реалізуємо ці дії на практиці:

1) Для об'єкту `Edit1` створюється опрацьовувач події `OnMouseDown`. В ньому перевіряється, чи натиснута саме ліва кнопка, і викликається метод `BeginDrag` з параметром `false`, що означає, що процес перетягування не почнеться, доки користувач

не перемістить мишу на деяку відстань. Це дозволить не плутати прості кліки на об'єкті з початком процесу перетягування

```
procedure TForm2.Edit1MouseDown (Sender: TObject;
Button: TMouseButton;
Shift: TShiftState;
X, Y: Integer);
Begin
If Button=mbLeft then
Edit1.BeginDrag(false);
End;
```

2) Для об'єкту `Panel1` створюється обробник події `OnDragOver` шляхом подвійного кліка на відповідному рядку панелі `Events` в Інспекторі об'єктів. У ньому перевіряється, чи є вихідний об'єкт полем введення `Edit1`. Можна виконати більш загальні перевірки, наприклад, встановити, чи належить вихідний об'єкт деякому класу і т.і. Результат перевірки заноситься в параметр `Accept`.

```
Procedure TForm2.Panel1DragOver (Sender, Source: TObject;
X,Y: Integer;
State: TDragState;
Var Accept: Boolean);
Begin
Accept:=Source is Edit1
End;
```

Можна виконати і більш загальну перевірку

```
Source is TEdit;
```

3) Панель повинна опрацювати ще одну подію `OnDragDrop`, генеровану, коли відбувається вставка перетягуваного об'єкта. В ньому проводиться підсумкова дія: в заголовок панелі записується текст, який знаходиться в текстовому полі.

```
Procedure TForm2.Panel1DragDrop (Sender, Source: TObject;
X,Y: Integer);
```

```
Begin
Panel1.Caption:=Edit1.Text
End;
```

Тепер можна запустити програму, навести вказівник на текстове поле (в ньому зберігається текст `Edit1`), натиснути ліву кнопку миші і, не відпускаючи її, перемістити курсор в напрямку до панелі. Вказівник при цьому прийме вигляд перекресленого кола, який означає, що в даному місці вставка перетягуваного об'єкту недопустима. Коли вказівник потрапить до області панелі, він матиме вигляд невеликої стрілки з квадратом. Тепер кнопку можна відпустити, і напис на панелі одразу зміниться на `Edit1`.

4) Переміщення кнопки по формі нагадує першу частину завдання. Для об'єкту `Button1` створюється опрацьовувач натискання кнопки миші, в якому починається процес перетягування кнопки:

```
Procedure TForm2.Button1MouseDown (Sender:
TObject;
Button: TMouseButton;
Shift: TShiftState;
X,Y: Integer);
Begin
If Button=mbLeft then
Button1.BeginDrag(false);
End;
```

Право приймати кнопку повинна мати сама форма. Для цього вона повинна так опрацьовувати подію `OnDragOver`, щоб дозволити прийом кнопок.

```
Procedure TForm2.FormDragOver(Sender,
Source:TObject;
X,Y: Integer;
State: TDragState;
Var Accept: Boolean);
Begin
Accept:=Source is Button
End;
```

Необхідно опрацювати також подію OnDragDrop. Для вихідної кнопки (параметр Source, приведений до типу TButton за допомогою операції As) задаються нові координати її місця розташування на формі (властивості Left і Top).

```
Procedure TForm2.FormDragDrop (Sender,
Source: TObject;
X,Y: Integer);
Begin
(Source as TButton).Left:=X;
(Source as TButton).Top:=Y;
End;
```

Тепер простими рухами миші кнопку можна вільно переміщувати по формі.

Контрольні питання

1) Вкажіть основні відмінності між проблемно-орієнтованими та об'єктно-орієнтованими ІК?

2) Який стиль взаємодії передбачає ООІК?

3) Що таке об'єкти-контейнери, об'єкти-пристрої, об'єкти-дані?

4) Які типи представлень об'єктів ви знаєте?

5) Які основні спільні проблеми для ГІК та ООІК?

6) Що таке властивості компонентів? Для чого вони використовуються?

7) Що таке методи в компонентах VCL? Як вони оголошуються?

8) Побудуйте ієрархію класів бібліотеки VCL.

9) Охарактеризуйте інтегроване середовище розроблення C++ Builder.

10) Як створюються однодокументний та багатодокументний інтерфейси?

11) Для чого використовується редактор форм C++ Builder?

12) Для чого використовується редактор меню C++ Builder?

13) Які візуальні компоненти є у VCL?

14) Які невізуальні компоненти є у VCL?

15) Назвіть компоненти інтерфейсу користувача в Delphi.

16) На чому заснована технологія виведення графічної інформації на екран в Delphi?

17) Охарактеризуйте класи `TCanvas`, `TGraphics`, `TPen`, `TBrush`, `TFont`.

18) Які підходи можливі при роботі з файлами в Delphi?

19) Яких дій вимагає технологія роботи з файлами в Delphi?

20) Порівняйте стандартний та об'єктний підхід до роботи з файлами в Delphi.

21) Як виконується друк у Delphi?

22) Для чого використовуються панелі `Additional`, `Win32`, `System` середовища Delphi?

Тестові питання

1) Як називається стиль взаємодії користувача з комп'ютером, який є засобом виведення середовища ГІК за межі простого представлення додатків і файлів у вигляді іконок на екрані?

1. Інтерфейс меню;
2. Графічний інтерфейс користувача;
3. Об'єктно-орієнтований інтерфейс користувача;
4. Інтерфейс командного рядка;
5. Інша відповідь.

2) Який стиль взаємодії притаманний ООІК?

1. Дія-об'єкт;
2. Об'єкт-дія;
3. Інтерфейс-дія;
4. Дія-інтерфейс;
5. Інша відповідь.

3) Які об'єкти надають користувачам інформацію і представляють будь-який тип інформації?

1. Об'єкти-дані;
2. Об'єкти-контейнери;
3. Об'єкти-пристрої;
4. Об'єкти-класи;
5. Інша відповідь.

4) Які об'єкти можуть зберігати і групувати будь-які об'єкти, представляючи їх вміст різними способами?

1. Об'єкти-дані;
2. Об'єкти-контейнери;
3. Об'єкти-пристрої;
4. Об'єкти-класи;
5. Інша відповідь.

5) Які специфікатори доступу, на Вашу думку, мають властивості компонентів?

1. Read specifier і write specifier;
2. Set;
3. Enumerations;
4. Public, private, protected;
5. Інша відповідь.

6) Які специфікатори доступу, на Вашу думку, мають методи в компонентах VCL?

1. Read specifier і write specifier;
2. Set;
3. Enumerations;
4. Public, private, protected;
5. Інша відповідь.

7) Які основні властивості форм доступні під час розробки і виконання програми?

1. OnActivate, OnClose, OnCreate, OnDestroy, OnDragDrop, OnMouseDown, OnPaint, OnResize, OnShow;
2. ActiveControl, AutoScroll, HorzScrollBar, VertScrollBar, BorderStyle, ClientWidth, ClientHeight, Font, FormStyle, HelpContext, Icon, Position,Visible, WindowState;
3. ActiveMDIChild, Canvas, ClientRect, Handle, ModalResult, Owner, Parent;
4. Show, ShowModal, BringToFront, Close, Print, ScrollInView, SetFocus;
5. Інша відповідь.

8) Яким чином можна виділити групу компонентів на формі (вибрати усі вірні відповіді)?

1. Натиснута клавіша Shift + клік миші на потрібних компонентах;
2. Натиснути кнопку миші та замкнути компоненти в рамку;
3. Обрати в головному меню пункт Edit|Select All;
4. Обрати в головному меню пункт View | Alignment Palette;
5. Інша відповідь.

9) Які дії слід виконати для створення підменю?

1. Натиснути клавішу Insert на пункті, над яким слід розташовувати новий пункт;
2. Натиснути клавішу Delete на пункті меню;
3. Вибрати в контекстному меню редактора Create Submenu або натиснути одночасно Ctrl і стрілку праворуч на клавіатурі;

4. Встановити для властивості Caption порожнього пункту меню значення "-" і натиснути Enter;

10) Фундаментальний клас, властивості і методи якого описують основні характеристики додатків Windows - це …?

1. TApplication;
2. TClipboard;
3. TForm;
4. TScreen;
5. Інша відповідь.

11) Які підходи можливі при роботі з файлами в Delphi 5?

1. Стандартний і нестандартний;
2. Об'єктний і процедурний;
3. Стандартний і об'єктний;
4. Нестандартний і процедурний;
5. Інша відповідь.

12) Вікно збереження файлу - це компонент …?

1. TOpenDialog;
2. TSaveDialog;
3. TFontDialog;
4. TColorDialog;
5. Інша відповідь.

13) Компонент TSpeedButton розташований на панелі…?

1. Additional;
2. System;
3. Win32;
4. Standard;
5. Інша відповідь.

14) Компонент TControlBar розташований на панелі…?

1. Additional;
2. System;
3. Win32;
4. Standard;
5. Інша відповідь.

Завдання для самоперевірки

1) Напишіть програму мовою програмування C++ з використанням ООІК GDI+ для реалізації наступної задачі. В центрі вікна намалюйте еліптичну діаграму. Діаграму розділіть на сектори 25%, 65% та 10% червоного, зеленого, блактиного кольорів та вказати по центру дуги кожного сектора відсотки. При всіх змінах розмірів вікна зображення діаграми повинно масштабуватись.

2) Напишіть програму мовою програмування C++ з використанням ООІК GDI+ для реалізації наступної задачі. Опишіть функцію, яка у прямокутнику будує еліптичну діаграму. Кількість секторів, їх розмір у відсотках та кольори також передаються аргументами виклику функції.

3) Напишіть програму мовою програмування C++ з використанням ООІК GDI+ для реалізації наступної задачі. Робочу область зафарбуйте спектром кольорів, починаючи від червоного і закінчуючи синім. Колір змінюйте в 4 етапи: на першому етапі, при максимальній червоній складовій, збільшуйте зелену складову, на другому, при максимальному зеленому, зменшуйте червону складову, далі, при максимальному зеленому, збільшуйте синю складову і, на останньому етапі, при максимальному синьому зменшіть зелену складову.

4) Напишіть програму мовою програмування C++ з використанням ООІК GDI+ для реалізації наступної задачі. Намалюйте світлофор, в якому колір "засвічується" при натисканні на нього лівою кнопкою миші. Одночасно може "горіти" лише один колір.

5) Напишіть програму мовою програмування C++ з використанням ООІК GDI+ для реалізації наступної задачі. У вікні додатку в 3 стовпчики виведіть привітання. У лівому і правому стовпцях розмір шрифта повинен зростати згори донизу, а в середньому - знизу догори. Кольори виведення повинні бути різні при кожному виведенні тексту. У випадку перекриття текстів різних стовпців текст не повинен зникати, причому правий стовпець може перекривати лівий, а середній - обидва стовпці.

6) Напишіть код мовою програмування C++ Builder для всіх пунктів меню вправи 4.

7) Створіть додаток мовою програмування C++ Builder, який при натисканні на кнопку відображає растрове зображення на головній формі.

8) Видаліть кнопки `Pause`, `Step Over` і `Trace Into` з панелі інструментів. Замість них додайте кнопки `Cut`, `Copy`, `Paste`. Поверніть панелі інструментів вигляд, який вона мала за замовчуванням. Завдання виконайте мовою програмування C++ Builder.

9) Розташуйте компонент `ListBox` на порожню форму і змініть його так, щоб він постійно займав всю робочу область форми. Завдання виконайте мовою програмування C++ Builder.

10)Мовою програмування C++ Builder створіть програму з вікном списку. Напишіть код для завантаження списку з текстового файлу перед відображенням головного вікна програми.

11)Засобами Delphi створіть додаток, який містить дві форми - на першій формі розташована кнопка, при натисканні на яку відображається друга форма.

12)Засобами Delphi створіть демонстраційний прототип додатку, схожого на Windows Notepad (Блокнот).

13)Засобами Delphi напишіть код для всіх пунктів меню демонстраційного прототипу завдання 12.

14)Засобами Delphi створіть додаток, який відображає растрове зображення в головній формі при натисканні на кнопку.

15)Засобами Delphi розташуйте компонент ListBox на порожню форму і змініть його так, щоб він постійно займав всю робочу область форми.