Тема 4. Розробка графічних додатків

1. VCL – надбудова над GDI

2. Об’єктний клас TCanvas

2.1. Основні властивості класу TCanvas

2.2. Основні методи класу TCanvas

3. Подія OnPaint

1. VCL – надбудова над GDI

GDI (Graphics Device Interface) – це та частина Windows, що забезпечує підтримку апаратно-незалежної графіки. C++Builder інкапсулює функції Windows GDI на різних рівнях [1]. Найбільш важливим тут є спосіб, за допомогою якого графічні компоненти представляють свої зображення на екрані монітора. При прямому виклику функції GDI необхідно передавати їм дескриптор контексту пристрою (device context handle), що задає обрані інструменти малювання – перо, пензель і шрифт. Після завершення роботи із графічними зображеннями обов'язково треба привести контекст пристрою у вихідний стан і тільки потім звільнитися від нього.

Типовий приклад малювання із застосовуванням стандартних функцій GDI може виглядати так:

```
void __fastcall TForm1::FormPaint(TObject *Sender)

{

HDC dc = GetDC(Handle); // дескриптор контексту

HPEN PenHandle, OldPenHandle;

PenHandle = CreatePen(PS_SOLID, 1, RGB(255, 0, 0));

OldPenHandle = SelectObject(dc, PenHandle);

HBRUSH BrushHandle, OldBrushHandle;

BrushHandle = CreateSolidBrush(RGB(255, 255, 0));

OldBrushHandle = SelectObject(dc, BrushHandle);

// жовтий еліпс, обведений червоним контуром

Ellipse(dc, 10, 10, 120, 100);
```

```
SelectObject(dc,OldBrushHandle) ;

DeleteObject(BrushHandle) ;

SelectObject(dc,OldPenHandle);

DeleteObject(PenHandle) ;

}
```

Замість того, щоб працювати із графікою на такому рівні деталізації, C++Builder надає простий і завершений інтерфейс за допомогою властивості Canvas (канва) графічних компонентів бібліотеки VCL. Ця властивість ініціалізує правильний контекст пристрою й звільняє його в потрібний час, коли припиняється малювання. Дескриптор контексту пристрою, над яким "побудована" канва, може бути потрібним для різних низькорівневих операцій. Він задається властивістю канви Handle. Клас TCanvas є обгорткою для HDC (HDC доступний через властивість Handle) і представляє більш високорівневий інтерфейс для роботи із графікою.

За аналогією з функціями Windows GDI канва має вкладені властивості, що представляють характеристики пера (Pen), пензеля (Brush) й шрифту (Font).

Єдине, що повинен зробити користувач, працюючи із графічними компонентами, – це визначити характеристики використовуваних інструментів малювання. З використанням Canvas зникає необхідність стежити за системними ресурсами при створенні, виборі й звільненні інструментів. Канва сама дбає про це.

У ряду компонентів з бібліотеки візуальних компонентів VCL є властивість Canvas (канва), яка надає простий шлях для малювання на них. TCanvas – це об’єктний клас, який інкапсулює графічні функції Windows. Канва не є компонентом, але вона входить як властивість у ряд компонентів, які повинні вміти намалювати себе й відобразити яку-небудь інформацію. Canvas (канва) надає простий шлях для малювання, наприклад, на компонентах TImage, TPaintBox, TForm.

Канва містить методи-надбудови над всіма основними функціями малювання GDI Windows. Всі геометричні фігури малюються поточним пером. Ті з них, які можна зафарбовувати, зафарбовуються за допомогою поточного пензля. Пензель і перо при цьому мають поточний колір. Крім того, можна малювати й поточково, одержавши доступ до кожного пикселя.

Клас TCanvas інкапсулює графічні методи: Draw, TextOut, Arc, Rectangle та ін. Використовуючи властивість Canvas, можна відтворювати на формі будь-які графічні об'єкти – картинки, многокутники, текст і т.п. без використання компонентів TImage, TShape і TLabel (тобто без використання додаткових ресурсів), однак при цьому треба обробляти подію OnPaint того об'єкта, на канві якого відбувається малювання.

Розглянемо приклад коду з використанням Canvas, який знову, як і в попередньому прикладі, малює жовтий еліпс, обведений червоним контуром, і наочно ілюструє, наскільки C++Builder спрощує програмування графіки.

```
void __fastcall TForm1::FormPaint(TObject *Sender)

{
```

```
Canvas->Pen->Color = clRed; // колір контуру

Canvas->Brush->Color = clYellow; // колір заливки

// жовтий еліпс, обведений червоним контуром

Canvas->Ellipse(10, 10, 120, 100);

}
```

Тут відбувалось малювання прямо на формі (можна було це вказати явно Form1-> Canvas->Ellipse(10, 10, 120, 100);).

Зазначимо, що тільки частина компонентів має властивість Canvas. Якщо треба малювати на компоненті, який не має такої властивості (наприклад, TPanel, TButton та інші), то здавалося б, що немає іншого виходу, ніж із застосовуванням стандартних функцій GDI. Але, виявляється, що за допомогою класу TControlCanvas можна приєднати канву до компонента, у якого її немає, і далі просто малювати так, як завжди.

Продемонструємо малювання еліпса на панелі (яка не має властивості Canvas) при натискання командної кнопки ***Button1***:

```
void __fastcall TForm1::Button1Click(TObject *Sender)

{

TControlCanvas *p=new TControlCanvas();

p->Control=(TControl*)Panel1;

p->Ellipse(10,10,120,100);

}
```

2. Об’єктний клас TCanvas

Canvas зв'язаний з усіма компонентами VCL, у яких є клієнтська частина, а також із класом TBitmap. Стандартні компоненти Windows такі як кнопки, списки й т.д. не мають властивості Canvas, тому що їх повністю малює Windows. Малювання на канві відбувається шляхом виклику відповідних функцій-членів (методів Draw, TextOut, Arc, Rectangle, Ellipse та ін.), а переключення режимів малювання відбувається шляхом модифікації властивостей класу TCanvas – Pen, Font, Brush, TextFlags та інших.

2.1. Основні властивості класу TCanvas

Brush – пензель, є об'єктом зі своїм набором властивостей:

Bitmap - картинка розміром строго 8´8, використовується для заповнення (заливання) області на екрані;

Color - колір заливання;

Style - визначений стиль заливання; ця властивість конкурує з властивістю Bitmap (та властивість з них, яка встановлена останньою, і буде визначати вигляд заливання;

Handle - дана властивість дає можливість використовувати пензель у прямих викликах процедур Windows API .

ClipRect – (тільки читання) прямокутник, на якому відбувається графічний вивід.

CopyMode – властивість визначає, яким чином буде відбуватися копіювання (метод CopyRect) на дану канву зображення з іншого місця: один до одного, з інверсією зображення й ін.

Font – шрифт, яким виводиться текст (методом TextOut).

Handle – використовується для прямих викликів Windows API.

Pen – перо, визначає вид ліній; як і пензель (Brush) є об'єктом з набором властивостей:

Color - колір лінії;

Handle - для прямих викликів Windows API;

Mode - режим виводу: проста лінія, з інвертуванням, з

виконанням виключаючого або та інші;

Style - стиль виводу: лінія, пунктир та інші;

Width - ширина лінії в точках;

PenPos - поточна позиція пера, перо рекомендується переміщати за допомогою методу MoveTo, а не прямою установкою даної властивості;

Pixels - двомірний масив елементів зображення (pixel), з його допомогою одержується доступ до кожної окремої точки зображення.

2.2. Основні методи класу TCanvas

- Методи для малювання найпростішої графіки:

MoveTo – встановлює поточну позицію пера (PenPos);

LineTo – малює пряму до заданої точки;

Rectangle – малює прямокутник із заданою діагоналлю;

Ellipse – малює еліпс, вписаний в заданий діагоналлю прямокутник;

Arc – малює частину кривою еліпсу;

Chord– малює частину кривою еліпсу, з'єднану хордою;

Pie – малює сектор еліпсу;

Polygon –малює замкнуту ламану;

PolyLine –малює незамкнуту ламану;

RoundRect – малює заокруглений прямокутник.

При промальовуванні ліній у цих методах використовуються перо (Pen) канви, а для заповнення внутрішніх областей – пензель (Brush).

- Методи для виводу тексту:

TextOut – виводить текстовий рядок (шрифтом (Font) канви);

TextRect – виводить текстовий рядок в прямокутнику (використовується шрифт (Font) канви);

TextHeight – задає його висоту;

TextWidth – задає його ширину.

- Методи для виводу картинок на канву:

Draw. Як параметри вказуються прямокутник і графічний об'єкт для виводу (це може бути TBitmap, TIcon або TMetafile);

StretchDraw. Від *Draw* відрізняється тим, що розтягує або стискає картинку так, щоб вона заповнила весь зазначений прямокутник.

- Методи для замальовування областей:

FillRect – замальовує прямокутник кольором і стилем пензля;

FloodFill – замальовує область довільної форми кольором і стилем пензля.

3. Подія OnPaint

При малюванні на канві форми або по **PaintBox** треба враховувати деякі особливості. Якщо вікно якогось іншого додатка перекриває вікно вашого додатка або, якщо малюнок тимчасово згортається, то зображення, намальоване на канві форми, псується. У компоненті **Image** цього не відбувається, оскільки в класі **TImage** уже передбачені всі необхідні дії, що здійснюють перемальовування зіпсованого зображення. А при малюванні на канві форми або інших віконних компонентів перемальовуванням повинен займатися сам розроблювач додатка. Якщо вікно було перекрито й зображення зіпсувалося, операційна система повідомляє додатку, що в оточенні щось змінилося й що додаток повинен почати відповідні дії. Якщо потрібне відновлення вікна, для нього генерується подія **OnPaint** [1]. В обробнику цієї події потрібно перемалювати зображення. Перемальовування може відбуватися різними способами залежно від задачі.

Припустимо на формі треба розмістити малюнок з деякого графічного файлу. Це можна зробити, помістивши у секцію private інтерфейсної частини класу TForm1 (файл Unit1.h) рядок:

```
// вказівник на графічний об'єкт з класу Graphics

Graphics:: TBitmap *pBitmap;
```

а у тіло конструктора класу TForm1 (файл Unit1.cpp) – рядок:

```
pBitmap = new Graphics:: TBitmap;

// обробник події OnClick малювання картинки

// (файл Unit1.cpp)

void __fastcall TForm1::FormClick(TObject *Sender)

{

if (OpenPictureDialogI->Execute())

pBitmap->LoadFromFile(OpenPictureDialogI->FileName);

Canvas->Draw(0,0,pBitmap);

}
```

Оскільки графічний об'єкт створився динамічно (оператор new), треба його знищити за допомогою оператора delete. Краще всього це зробити за допомогою деструктора класу (секція public).

Файл Unit1.h:

__fastcall ~TForm1 (void);

Файл Unit1.cpp:

__fastcall ~TForm1::TForm1 (void)

{

delete pBitmap;

}

// обробник події OnPaint (файл Unit1.cpp)

void __fastcall TForm1::FormPaint(TObject *Sender)

{

if (pBitmap != NULL) Canvas->Draw(0,0,pBitmap);

}

Наведений вище обробник події **OnPaint** перемальовує все зображення, хоча, може бути, зіпсована тільки частина його. При великих зображеннях це може значно уповільнювати перемальовування. Перемальовування можна істотно прискорити, якщо перемальовувати тільки зіпсовану область канви.

У канви є властивість **ClipRect** типу TRect, що у момент обробки події **OnPaint** указує на область, що підлягає перемальовуванню. Тому більше ефективним буде обробник:

void_fastcall TForm1::FormPaint(TObject *Sender)

{

if (pBitmap != NULL)Canvas->CopyRect

(Canvas->ClipRect, pBitmap->Canvas, Canvas->ClipRect);

}

Він перемальовує тільки прямокутну область **ClipRect,** яка зіпсована.