1. Робота з графікою в Visual C# 2010 Express Edition.

Мета: **:** Засвоїти методику та виробити практичні навички у проектуванні та створенні програм з графічним інтерфейсом GDI+.

Теоретичні відомості.

Нова концепція графічного інтерфейсу GDI+( **Graphics Device Interface**) дещо відрізняється від концепці «класичного» графічного інтерфейсу GDI. Для того щоб програма міг що-небудь намалювати у вікні, вона повинна, перш за все, отримати або створити для цього об'єкт класу Graphics. Користуючись властивостями і методами цього об'єкта, програма може малювати у вікні різні фігури або текстові рядки.

***Ідентифікатор вікна Handle і об'єкт Graphics.***

Перш ніж намалювати у вікні нашої програми, потрібно мати для цього вікна об'єкт класу Graphics. Кожне вікно має свій ідентифікатор (handle). Знаючи ідентифікатор вікна, можна легко отримати пов'язаний з цим вікном контекст відображення. Програми Microsoft. NET Framework можуть отримати ідентифікатор форми або будь-якого іншого елемента керування за допомогою властивості Handle. Програма отримує ідентифікатор вікна форми *Form1* за допомогою властивості *this.Handle.* Знаючи ідентифікатор вікна, за допомогою методу Graphics.*FromHwnd* неважко отримати потрібний об'єкт класу *Graphics*:

*Graphics g = Graphics.FromHwnd (this.Handle);*

 Програміст, що створює додаток GDI +, має потребу в інструментах для малювання. Створюється кисть як об'єкт класу SolidBrush:

*SolidBrush redBrush = new SolidBrush (Color. Red);*

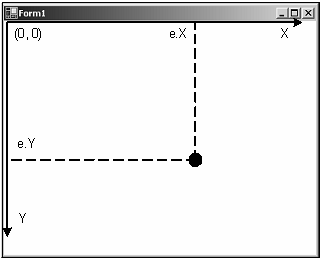
За допомогою цієї кисті можна малювати замкнуті геометричні фігури, зафарбовані заданим кольором. Через єдиний параметр передається конструктор класу SolidBrush колір кисті Color.Red. Таким чином, малюється пензлем червоного кольору.

У класі Graphics є безліч різних методів, призначених для малювання самих різних геометричних фігур, таких як лінії, прямокутники, овали та кола, багатокутники, криві Безьє і т.д. Але в цьому класі немає метода, за допомогою якого можна було б намалювати одну єдину точку. Замість крапки можна намалювати зафарбований квадрат з шириною боку, рівним 1 пікселу. Це виконується за допомогою методу *FillRectangle*:

*g. FillRectangle (redBrush, e. X, e. Y, 1, 1);*

Метод *FillRectangle* викликається для об'єкта g класу Graphics. В якості першого параметра методу FillRectangle передається кисть redBrush, яку потрібно використовуваті для малювання. Кисть потрібна і для інших методів класу Graphics, призначення для малювання геометричних фігур. Другий и Третій параметр методу *FillRectangle* задають координати, в якіх буде намальованій квадрат. Початок системи координат при цьому знаходиться в лівому верхньому кутку вікна, для якого був отриманий об'єкт Graphics.

Вісь X в цій системі координат, прийнятій за замовчуванням, спрямована зліва направо, а вісь Y - зверху вниз.

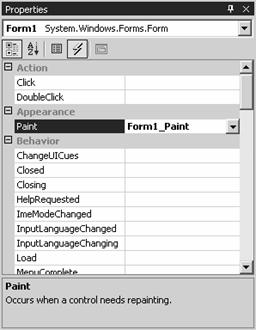


Останні два параметри методу *FillRectangle* задають, ширину і висоту прямокутника.

Призначення параметрів методу FillEllipse, призначеного для малювання зафарбованих еліпсів, аналогічно призначенню параметрів методу FillRectangle. При цьому два останніх параметра задають, відповідно, ширину і висоту області, займаної еліпсом.

g.FillEllipse (redBrush, eX, eY, 10, 10);

Для форм  передбачений зручний об'єктно-орієнтована спосіб, що дозволяє при необхідності перемальовувати вікно форми в будь-який момент часу. Коли вся клієнтська область вікна форми або частина цієї області вимагає перемальовування, формі передається подія Paint. Все, що потрібно від програміста, це створити обробник даної події, наповнивши його необхідною функціональністю.



Ось в якому вигляді буде створений обробник події Paint:

private void Form1\_Paint (object sender,   
*System.Windows.Forms.PaintEventArgs e)   
{   
}*

Обробникові Form 1\_ Paint передаються два параметри.

Через перший параметр передається посилання на об'єкт. Що ж стосується другого параметра, то через нього передається посилання на об'єкт класу PaintEventArgs. Цей об'єкт має дві властивості, доступних тільки для читання - Graphics і ClipRectangle.

Клас Graphics - є контекст відображення, необхідний для малювання тексту і геометричних фігур. Обробник події Paint отримує контекст відображення через свої параметри, тому програмісту не потрібно визначати його спеціальним чином.

Через властивість ClipRectangle передаються межі області, яку повинен перемалювати обробник події Paint.

Обробник події Paint може ігнорувати властивість ClipRectangle, перемальовуючи вміст вікна повністю. Однак процес перемальовування вмісту вікна можна помітно прискорити, якщо перемальовувати не все вікно, а тільки область, описану властивістю ClipRectangle. Прискорення буде особливо помітним, якщо у вікні намальовано багато тексту і геометричних фігур.

public string text;

…..

public Form1 ()

{  
 / / Required for Windows Form Designer support

/ /

InitializeComponent ();

/ /

/ / TODO: Add any constructor code after InitializeComponent call

/ /

text = "Обробка події Paint";

}  
Потрібно поміняти вихідний текст обробника події Form1\_Paint наступним чином:

private void Form1\_Paint (object sender, System.Windows.Forms.PaintEventArgs e)

{  
Graphics g = e.Graphics;

g.Clear (Color.White);

g.DrawString (text, new Font ("Helvetica", 15), Brushes.Black, 0, 0);

g.DrawRectangle (new Pen (Brushes.Black, 2), 10, 30, 200, 100);

g.DrawEllipse (new Pen (Brushes.Black, 2), 150, 120, 100, 130);

}

Тут в тілі обробника Form1\_Paint визначили локальну змінну g класу Graphics, призначену для зберігання контексту відображення. Ця змінна ініціалізується за допомогою значення, отриманого з властивості Graphics першого параметра обробника Form1\_Paint:

Graphics g = e. Graphics;

Отримавши контекст відображення, обробник події Paint може малювати у відповідному вікні все, що завгодно. Спочатку зафарбовується вікно білим кольором, викликаючи для цього метод Clear, визначений в класі Graphics:

g. Clear (Color. White);

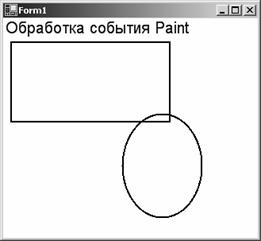
Таким способом можна зафарбувати фон, колір якого заданий для форми у властивості BackColor. Методи DrawString, DrawRectangle і DrawEllipse, також визначені в класі Graphics:

g.DrawString (text, new Font ("Helvetica", 15), Brushes.Black, 0, 0);

g.DrawRectangle (new Pen (Brushes.Black, 2), 10, 30, 200, 100);

g.DrawEllipse (new Pen (Brushes.Black, 2), 150, 120, 100, 130);

Перший з них малює текстовий рядок у верхній частині вікна, а два інших - прямокутник і еліпс, відповідно.



Метод DrawLine малює лінію, що сполучає дві точки з заданими координатами:

public void DrawLine (Pen, Point, Point);

public void DrawLine (Pen, PointF PointF;

public void DrawLine (Pen, int, int, int, int);

public void DrawLine (Pen, float, float, float, float);

Перший параметр задає інструмент для малювання лінії - перо. Пір'я створюються як об'єкти класу Pen, наприклад:

Pen p = new Pen (Brushes. Black, 2);

Створюється чорне перо товщиною 2 пікселя. Створюючи пероможна вибрати його колір, товщину і тип лінії, а також інші атрибути. Інші параметри перевантажених методів DrawLine задають координати з'єднуючих точок. Ці координати можуть бути задані як об'єкти класу Point і PointF, а також у вигляді цілих чисел і чисел з плаваючою крапкою.

У класах Point і PointF визначені властивості X і Y, які визначають, відповідно, координати точки по горизонтальній і вертикальної осі. При цьому в класі Point ці властивості мають цілочисельні значення, а в класі PointF - значення з плаваючою крапкою.

Третій і четвертий варіант методу DrawLine дозволяє задавати координати з'єднуються точок у вигляді двох пар чисел. Перша пара визначає координати першої точки по горизонтальній і вертикальній осі, а друга - координати другої точки по цих же осях. Різниця між третім і четвертим методом полягає у використанні координат різних типів (цілочисельних int і з плаваючою крапкою float).

private void Form 1\_ Paint (object sender, System. Windows. Forms. PaintEventArgs e)

{

Graphics g = e. Graphics;

g. Clear (Color. White);

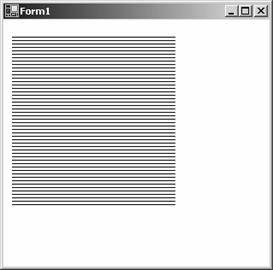
for (int i = 0; i <50; i + +)

{

g. DrawLine (new Pen (Brushes. Black, 1), 10, 4 \* i + 20, 200, 4 \* i + 20);

}

}  
Тут викликається метод DrawLine в циклі, Малюється 50 горизонтальних ліній.



Викликавши один раз метод DrawLines, можна намалювати відразу кілька прямих ліній, з'єднаних між собою. Іншими словами, метод DrawLines дозволяє з'єднати між собою декілька точок. Координати цих точок по горизонтальній і вертикальної осі передаються методом через масив класу Point або PointF

Pen pen = new Pen (Color.Black, 2);

Point [] points = new Point [50];

public Form1 ()

{

InitializeComponent ();

for (int i = 0; i <20; i + +)

{

int xPos;

if (i% 2 == 0)

{

xPos = 10;

}

Else

{

xPos = 400;

}

points [i] = new Point (xPos, 10 \* i);

}

}

Координати точок по горизонтальній осі залежать від того, чи є значення змінної циклу i парних або непарних. Малювання ліній виконується за один виклик методу DrawLines під час обробки події Paint:

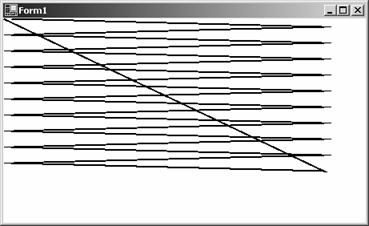
private void Form1\_Paint (object sender, System.Windows.Forms.PaintEventArgs e)

{

Graphics g = e.Graphics;

g.Clear (Color.White);

g.DrawLines (pen, points);

}  


Метод DrawRectangle дозволяє малювати прямокутники, задані координатою верхнього лівого кута, а також шириною і висотою. У бібліотеці класів. NET Frameworks є три перевантажених варіанту цього методу:  
public void DrawRectangle (Pen, Rectangle);  
public void DrawRectangle (Pen, int, int, int, int);  
public void DrawRectangle (Pen, float, float, float, float);

В якості першого параметра цим методам передається перо класу Pen. Інші параметри задають розташування і розміри прямокутника.

Клас Rectangle використовується для опису розташування і розмірів прямокутної області. Властивості X та Y цього класу задають координати верхнього лівого кута прямокутної області, відповідно, по горизонтальній і вертикальної осі координат. Властивості Width і Height, зберігають, відповідно, ширину і висоту прямокутної області. У класі Rectangle визначені й інші властивості, а також методи. Детальний опис цього класу Ви знайдете в документації.

Варіанти методу DrawRectangle з п'ятьма параметрами дозволяють задавати розташування і розміри прямокутника у вигляді цілих чисел, а також у вигляді числі з плаваючою крапкою. Другий і третій параметр задає розташування верхнього лівого кута по горизонтальній і вертикальній осі координат, відповідно, а четвертий і п'ятий - ширину і висоту прямокутника.

Pen myPen = new Pen (Color.Black, 2);

Rectangle [] myRectsArray =

{  
new Rectangle (10, 10, 200, 200),  
new Rectangle (20, 20, 180, 180),  
new Rectangle (30, 30, 160, 160),  
new Rectangle (40, 40, 140, 140)  
};

Метод DrawRectangles викликається в тілі обробника події Paint, вихідний текст якого наведено нижче:

private void Form1\_Paint (object sender, System.Windows.Forms.PaintEventArgs e)

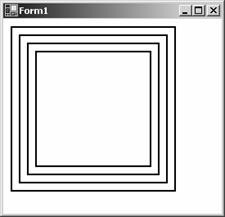
{

Graphics g = e.Graphics;

g.Clear (Color.White);

g.DrawRectangles (myPen, myRectsArray);

}



Багатокутник

Метод DrawPolygon допоможе Вам, коли потрібно намалювати багатокутник, заданий своїми вершинами.

Передбачено два варіанти цього методу:

public void DrawPolygon (Pen, Point[]);   
public void DrawPolygon (Pen, PointF[]);

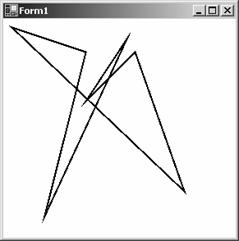
У першому випадку методу DrawPolygon через другий параметр передається масив точок класу Point, в якому координати точок задані цілими числами, а в другому — масив класу PointF , де координати з’єднувальних точок задаються у вигляді чисел із плаваючою десятковою точкою.

Для демонстрації можливостей методу DrawPolygon створено додаток DrawPolygon App. В ньому визначено перо myPen і масив точок myPoints :

Pen myPen = new Pen(Color.Black, 2);   
  
Point[] myPoints =   
{   
new Point(10, 10),   
new Point(100, 40),   
new Point(50, 240),   
new Point(150, 24),   
new Point(100, 100),   
new Point(160, 40),   
new Point(220, 210)   
};

Обробник події Form1\_Paint з’єднує ці точки разом, викликаючи метод DrawPolygon :

private void Form1\_Paint(object sender, System.Windows.Forms.PaintEventArgs e)   
{   
Graphics g=e.Graphics;   
g.Clear(Color.White);   
g.DrawPolygon(myPen, myPoints);   
}



**Еліпс**

Метод DrawEllipse рисує еліпс, вписаний у прямокутну область, розміщення і розміри якої передаються йому в якості параметрів.

Передбачено чотири перевантажених варіанти методу DrawEllipse:

public void DrawEllipse (Pen, Rectangle);   
public void DrawEllipse (Pen, RectangleF);   
public void DrawEllipse (Pen, int, int, int, int);   
public void DrawEllipse (Pen, float, float, float, float);

Ці методи відрізняються тільки способом, за допомогою якого описується розташування і розміри прямокутної області, в яку вписаний еліпс. Можна задавати розташування і розміри цієї області у вигляді розглянутих раніше об'єктів класу Rectangle, RectangleF, а також у вигляді цілих чисел або числі з плаваючою крапкою.

Сегмент еліпса

За допомогою методу Draw Arc програма може намалювати сегмент еліпса. Сегмент задається за допомогою координат прямокутної області, в яку вписаний еліпс, а також двох кутів, відлічуваних в напрямку проти годинникової стрілки. Перший кут Angle 1 задає розташування одного кінця сегмента, а другий Angle 2 - розташування іншого кінця сегмента (рис. 10-13).

Передбачено чотири перевантажених варіанти методу Draw Arc :

public void DrawArc (Pen, Rectangle, float, float);   
public void DrawArc (Pen, RectangleF, float, float);   
public void DrawArc (Pen, int, int, int, int, int, int);   
public void DrawArc (Pen, float, float, float, float, float, float);

Перший параметр методу Draw Arc визначає перо, з допомогою якого буде намальований сегмент. Останні два параметри задають кути Angle 1 і Angle 2 відповідно до рис. 10-13. Розташування та розміри прямокутної області передаються методу Draw Arc аналогічно тому, як це робиться для розглянутого вище методу DrawEllipse.

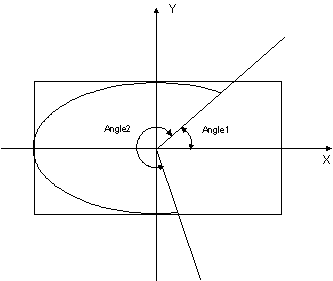
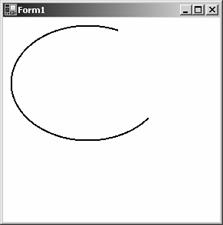


Рис. 10-13. Кути і прямокутник, що задають сегмент еліпса

Для малювання сегмента еліпса ми створили додаток DrawArcApp. Вся робота з малювання виконується всередині обробника події Form1\_Paint:

private void Form1\_Paint(object sender,   
System.Windows.Forms.PaintEventArgs e)   
{   
Pen myPen = new Pen(Color.Black, 2);   
  
Graphics g=e.Graphics;   
g.Clear(Color.White);   
g.DrawArc(myPen, 10, 10, 200, 150, 30, 270);   
}



**Криві Безьє**

Сплайн являє собою криву лінію, що сполучає між собою кілька точок. Крива Безье, що представляє собою одну з різновидів сплайна, задається чотирма точками. Дві з них - початкова і кінцева, а дві інші - керуючі. Крива Безье проходить через початкову та кінцеву точки, а управляючі точки задають вигини кривої лінії. Ті з Вас, хто коли-небудь працював з векторними графічними редакторами, наприклад, з редактором Corel Draw, знають про існування кривих Безьє і керуючих точок.

Для малювання кривих Безьє є два перевантажених набору методів DrawBezier і DrawBeziers:

public void DrawBezier (Pen, Point, Point, Point, Point);   
public void DrawBezier (Pen, PointF, PointF, PointF, PointF);   
public void DrawBezier (Pen, float, float, float, float, float, float,   
float, float);

public void DrawBeziers (Pen, Point[]);   
public void DrawBeziers (Pen, PointF[]);

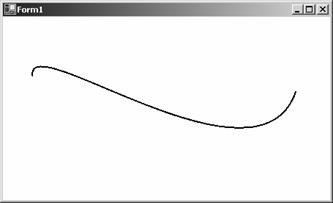
У всіх цих методах перший параметр задає перо, яке буде використане для малювання. Інші параметри задають координати початкової, кінцевої і керуючих точок.

Що стосується методу DrawBeziers, то він дозволяє задавати координати точок у вигляді масивів, що може бути зручно в деяких випадках.

У додатку DrawBezierApp намальовано криву Безьє в обробнику події Paint:

private void Form 1\_ Paint ( object sender ,   
System . Windows . Forms . PaintEventArgs e )   
{   
Pen myPen = new Pen ( Color . Black , 2);   
  
PointF startPt = new PointF (40.0 F , 80.0 F );   
PointF control 1 Pt = new PointF (30.0 F , 10.0 F );   
PointF control 2 Pt = new PointF (350.0 F , 250.0 F );   
PointF endPt = new PointF (400.0 F , 100.0 F );   
  
PointF [] myBezierPoints =   
{   
startPt ,   
control 1 Pt ,   
control 2 Pt ,   
endPt   
};   
Graphics g = e . Graphics ;   
g . Clear ( Color . White );   
g . DrawBeziers ( myPen , myBezierPoints );   
}

Створено початкову та кінцеву точки startPt і endPt, через які проходить крива, а також управляючі точки control1Pt і control2Pt. Координати всіх точок передаються методу DrawBeziers через масив myBezierPoints. Керуючі точки згинають лінію, як би притягаючи її до себе



**Зафарбовані фігури**

У класі Graphics визначено ряд методів, призначених для малювання зафарбованих фігур. Імена деяких з цих методів, які мають префікс Fill, перераховано в табл. 10 2.  
Таблиця 10-2. Методи для малювання зафарбованих фігур

|  |  |
| --- | --- |
| Метод | Опис |
| FillRectangle | Малювання зафарбованого прямокутника |
| FillRectangles | Малювання множини зафарбованих прямокутників |
| FillPolygon | Малювання зафарбованого багатокутника |
| FillEllipse | Малювання зафарбованого еліпса |
| FillPie | Малювання зафарбованого сегмента еліпса |
| FillClosedCurve | Малювання зафарбованого сплайна |
| Fill Region | Малювання зафарбованої області типу Region |

Є дві відмінності методів з префіксом Fill від однойменних методів з префіксом Draw. Перш за все, методи з префіксом Fill малюють зафарбовані фігури, а методи з префіксом Draw - незафарбовані. Крім цього, в якості першого параметра методам з префіксом Fill передається не перо класу Pen, а кисть класу Brush.  
Приклад:

SolidBrush redBrush = new SolidBrush ( Color . Red );   
g . FillEllipse ( redBrush , 50, 50, 100, 110);

Тут спочатку створюємо кисть червоного кольору як об'єкт класу SolidBrush. Ця кисть потім передається методу FillEllipse в якості першого параметра. Інші параметри методу FillEllipse задають розташування і розміри прямокутника, в який буде вписаний еліпс.

**Інструменти для малювання**

Піря

Піря використовуються для малювання ліній і простих геометричних фігур та створюються як об’єкти класу Pen . Відповідні конструктори :

public Pen(Color);   
public Pen(Color, float);   
public Pen(Brush);   
public Pen(Brush, float);

Перший із цих конструкторів створює перо заданного кольору. Колір задається за допомогою об'єкту класу Color. Другий конструктор дозволяє додатково задати товщину пера.

Третій і четвертий конструктор створюють перо на основі кисті, причому в четвертому конструкторі можна вказати товщину створюваного пера. Про кистях ми розповімо трохи пізніше в цьому розділі.

Після того як перо створено, програма може визначити його атрибути за допомогою властивостей класу Pen. Деякі з цих властивостей перераховані в табл. 10-7.

Таблиця 10-7. властивості пера

|  |  |
| --- | --- |
| Властивість | Опис |
| Alignment | Вирівнювання пера |
| Width | Ширина линії |
| Brush | Кисть, що використовується пером |
| Color | Колір пера |
| DashStyle | Стиль пунктирних і штрих-пунктирних ліній |
| DashCup | Вид точок і штріхов пунктирних і штрих-пунктирних ліній |
| DashOffset | Відстань від початку лінії до початку штриха |
| DashPattern | Масив шаблонів для створення довільних штрихів і пробілів штрихових і штрих-пунктирних ліній |
| StartCup  EndCup | Стиль кінців ліній |
| LineCap | Форми кінців ліній |
| LineJoin | Стиль зєднання кінців двох різних ліній |
| MiterLimit | Гранична товщина в області з'єднання гострих ліній |

Встановлюючи значення властивості Color і Width, додаток може змінити, відповідно, колір і ширину лінії, що малюються пером.

Якщо треба намалювати пунктирну або штрих-пунктирну лінію, програма має задати необхідне значення для властивості DashStyle. При цьому допускається змінювати вигляд крапок і тире пунктирних і штрих-пунктирних ліній (властивість DashCup), задавати відстань від початку лінії до початку штриха (властивість DashOffset) або навіть зовсім задати довільний вид для штрихів і розділяють ці штрихи прогалин (властивість DashPattern).

При необхідності змінити зовнішній вигляд кінців ліній використовуйте властивості StartCup і EndCup, що задають стиль кінців ліній. Властивість LineCap визначає форму кінців ліній.

У додатку PenApp показано способи зміни товщини лінії, стилю пунктирною і штрих-пунктирною лінії, а також стилю кінців ліній. Вихідний текст обробника події Form1\_Paint, який малює лінії різних типів і стилів:

using System.Drawing.Drawing2D;   
...   
private void Form 1\_ Paint ( object sender , System . Windows . Forms . PaintEventArgs e )   
{   
Graphics g = e . Graphics ;   
g . Clear ( Color . White );   
  
int x =10;   
int y =10;   
Pen myPen = new Pen ( Color . Black , 1);   
  
g . DrawLine ( myPen , x , y , 200, y );   
  
y += 15;   
myPen . Width = 2;   
g . DrawLine ( myPen , x , y , 200, y );   
  
y += 15;   
myPen . Width = 3;   
g . DrawLine ( myPen , x , y , 200, y );   
  
y += 15;   
myPen . Width = 5;   
g . DrawLine ( myPen , x , y , 200, y );   
  
y += 15;   
myPen . Width = 10;   
g . DrawLine ( myPen , x , y , 200, y );   
  
y += 15;   
myPen . Width = 3;   
myPen . DashStyle = DashStyle . Dash ;   
g . DrawLine ( myPen , x , y , 200, y );   
  
y += 15;   
myPen . Width = 3;   
myPen . DashStyle = DashStyle . DashDot ;   
g . DrawLine ( myPen , x , y , 200, y );   
  
y += 15;   
myPen . Width = 3;   
myPen . DashStyle = DashStyle . DashDotDot ;   
g . DrawLine ( myPen , x , y , 200, y );   
  
y += 15;   
myPen . Width = 3;   
myPen . DashStyle = DashStyle . Dot ;   
g . DrawLine ( myPen , x , y , 200, y );   
  
y += 15;   
myPen . Width = 5;   
myPen . DashStyle = DashStyle . Solid ;   
myPen . StartCap = LineCap . ArrowAnchor ;   
myPen . EndCap = LineCap . DiamondAnchor ;   
g . DrawLine ( myPen , x , y , 200, y );   
  
y += 15;   
myPen . Width = 5;   
myPen . StartCap = LineCap . Round ;   
myPen . EndCap = LineCap . RoundAnchor ;   
g . DrawLine ( myPen , x , y , 200, y );   
  
y += 15;   
myPen . Width = 5;   
myPen . StartCap = LineCap . Square ;   
myPen . EndCap = LineCap . SquareAnchor ;   
g . DrawLine ( myPen , x , y , 200, y );   
  
y += 15;   
myPen . Width = 5;   
myPen . StartCap = LineCap . Triangle ;   
myPen . EndCap = LineCap . Flat ;   
g . DrawLine ( myPen , x , y , 200, y );   
}

На початку своєї роботи метод Form1\_Paint отримує посилання на обєкт класу Graphics , тобто контекст відображення:

Graphics g = e . Graphics ;

Використовуючи отриманий контекст відображення, метод Form1\_Paint зафарбовує вікно додатка білим кольором, а тоді малює чорну лінію товщиною 1 піксель:

g.Clear(Color.White);   
int x=10;   
int y=10;   
Pen myPen = new Pen(Color.Black, 1);   
  
g.DrawLine(myPen, x, y, 200, y);

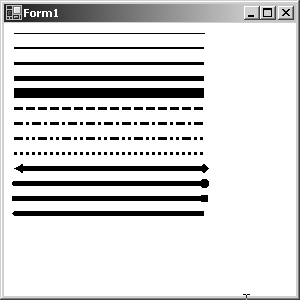
Змінюємо властивість Width створеного пера myPen , і знову малюємо лінію з невеликим зміщенням по вертикалі:

y += 15;   
myPen.Width = 2;   
g.DrawLine(myPen, x, y, 200, y);

Після цього малюємо ще декілька ліній збільшуючої товщини:

y += 15;   
myPen.Width = 3;   
myPen.DashStyle = DashStyle.Dash;   
g.DrawLine(myPen, x, y, 200, y);   
  
y += 15;   
myPen.Width = 3;   
myPen.DashStyle = DashStyle.DashDot;   
g.DrawLine(myPen, x, y, 200, y);   
  
y += 15;   
myPen.Width = 3;   
myPen.DashStyle = DashStyle.DashDotDot;   
g.DrawLine(myPen, x, y, 200, y);   
  
y += 15;   
myPen.Width = 3;   
myPen.DashStyle = DashStyle.Dot;   
g.DrawLine(myPen, x, y, 200, y);

Результат виконання цих операцій



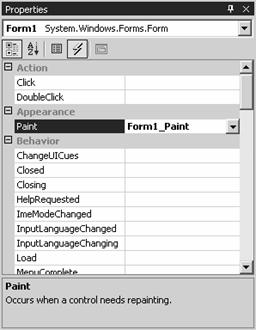
**Приклад виконання лабораторної роботи.**

**Завдання.** Спроектувати та розробити програму, яка б виводила радіо елемент та за допомогою клавіатури рухати його по формі.

Приклад на основі зображення «смайлик».

Хід виконання прикладу:

1. Створити Windows Forms проект.
2. В події Paint пишемо код для малювання елементу.



1. В події KeyPress пишемо код для роботи з клавіатурою.

**Код програми:**

Клас Form1.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

namespace WindowsFormsApplication1

{

public partial class Form1 : Form

{

String text = "Смайлик";

int p = 0;

int x, y;

public Form1()

{

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

x = 250; y = 280;

}

private void smile(PaintEventArgs e, Graphics g,int x, int y)

{

g.Clear(Color.White);

g.DrawString(text, new Font("Helvetica", 15), Brushes.Black, 250, 50);

g.DrawEllipse(new Pen(Brushes.Black, 2), x, y, 100, 100);

g.DrawLine(new Pen(Brushes.Black, 2), x + 30, y + 70, x + 70, y + 70);

g.DrawEllipse(new Pen(Brushes.Black, 2), x + 20, y + 30, 10, 10);

g.DrawEllipse(new Pen(Brushes.Black, 2), x + 60, y + 30, 10, 10);

}

private void smileMove(Graphics g, int x, int y)

{

g.Clear(Color.White);

g.DrawString(text, new Font("Helvetica", 15), Brushes.Black, 250, 50);

g.DrawEllipse(new Pen(Brushes.Black, 2), x, y, 100,100);

g.DrawLine(new Pen(Brushes.Black, 2), x + 30, y + 70, x + 70, y + 70);

g.DrawEllipse(new Pen(Brushes.Black, 2), x + 20, y + 30, 10, 10);

g.DrawEllipse(new Pen(Brushes.Black, 2), x + 60, y + 30, 10, 10);

}

private void Form1\_Paint(object sender, PaintEventArgs e)

{

Graphics g = e.Graphics;

smile(e, g,x,y);

}

private void Form1\_KeyPress(object sender, KeyPressEventArgs e)

{

Graphics g = Graphics.FromHwnd(this.Handle);

g.Clear(Color.White);

if (e.KeyChar == 'w')

{

this.smileMove(g, x, y);

y = y - 1;

}

else if (e.KeyChar == 's')

{

this.smileMove(g, x, y);

y = y + 1;

}

else if (e.KeyChar == 'd')

{

this.smileMove(g, x, y);

x = x + 1;

}

else if (e.KeyChar == 'a')

{

this.smileMove(g, x, y);

x = x - 1;

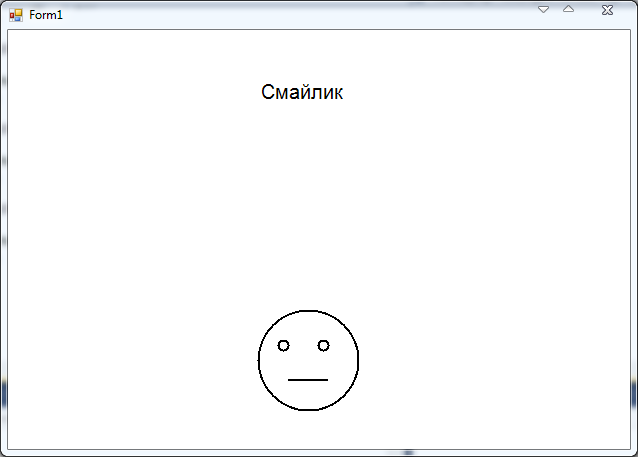
}

}

}

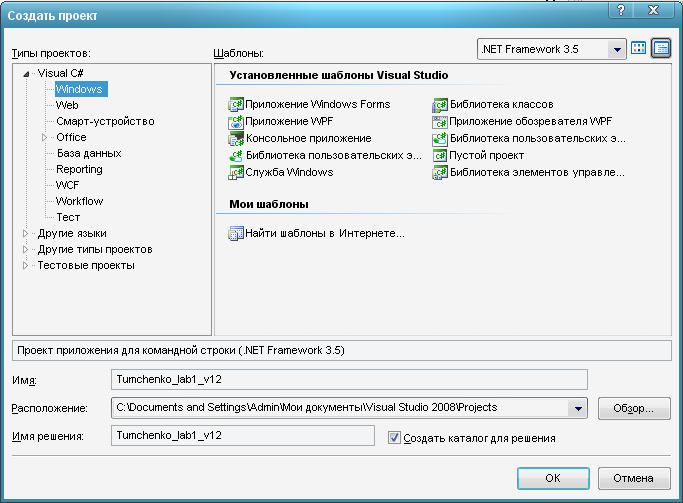
}

Результат виконання програми



**Хід роботи.**

1. Ознайомитися з теоретичними відомостями та прикладом виконання лабораторної роботи.
2. За номером у журналі вибрати власний варіант.
3. Розробити блок схему програми чи UML діаграми.
4. Запустити Visual Studio 2003/2005/2008. Вибрати в меню **Файл/Создать/Проект,** у вікні, що відкриється вибрати тип проекту – Windows Forms проект і ввести назву яка відповідає призвіщу студента, його варіанту і номеру лабораторної роботи.



1. Реалізувати програму згідно завдання і варіанту.
2. Розробити звіт. Звіт має включати, номер і назву лабораторної роботи, мету, теоретичні відомості, задачу, розробку структури даних (опис змінних та їх тип), блок-схему чи UML діаграми (діаграму класів), код програми, результат виконання програми та висновок.
3. Захист лабораторної роботи.

**Варіанти завдань.**

|  |  |  |  |
| --- | --- | --- | --- |
| *№ варіанту* | *Завдання* | *№ варіанту* | *Завдання* |
| 1 | Перехід | 16 | Поперечний |
| 2 | Омічеський | 17 | З одним переходом |
| 3 | Діод | 18 | Польовий транзистор |
| 4 | Тунельний діод | 19 | Стабілітрон |
| 5 | Світло діод | 20 | *Ламповий діод* |
| 6 | Ламповий тріод | 21 | Ламповий тедрот |
| 7 | Симістор | 22 | Керуючий переключатель |
| 8 | Біполярний транзистор | 23 | Керуючий виправляч |
| 9 | Концентратор ліній | 24 | **Діністор** |
| 10 | Виправляч | 25 | Керуючий випрямляч |
| 11 | Світовий індикатор | 26 | Люмінесцентна лампа |
| 12 | Лампа | 27 | Ламповий пентод |
| 13 | Тріод | 28 | Триод |
| 14 | Транзистор | 29 | Варактор |
| 16 | Симістор | 30 | Оміческий |

**Вимоги до звітів.**

Варіант Практичної роботи обирається згідно списку в журналі.

З усіх чотирьох боків аркуша повинні бути залишені поля розміром 20 мм, заповненість сторінки має становити 75% від її загального обсягу. Текст набирають на комп'ютері, шрифтом ґарнітурою Times New Roman. Заголовок розділу, підрозділу від попереднього тексту відокремлюють трьома інтервалами. Відступ від назви підрозділу (пункту, підпункту) до наступного тексту повинен складати півтора інтервала. Назви розділів та підрозділів для зручності читання потрібно виділяти напівжи-рним шрифтом, текст звіту вирівнювати по ширені, а визначення понять у тексті та термінів – курсивом.

Звіт має включати, номер і назву лабораторної роботи, мету, теоретичні відомості, задачу, розробку структури даних (опис змінних та їх тип), блок-схему чи UML діаграми (діаграму класів, діаграму прецедентів, діаграма послідовності), код програми, результат виконання програми та висновок.

Звіт здається скріпленими скріпкою у фай лику.

**Оцінювання практичної роботи.**

Практична робота оцінюється на **9-10 балів** коли студент: повністю зробив програму згідно варіанту, та оформив звіт згідно вимог; при захисті звіту дає повну, розгорнуту, вичерпну відповідь на питання; здатний здійснювати - порівняльний аналіз різних теорій, концепцій, підходів та самостійно робить логічні висновки й узагальнення; здатний висловлювати та аргументувати власне ставлення до альтернативних поглядів на дану тему.

Практична робота оцінюється на **7-8 балів** коли студент: повністю зробив програму згідно варіанту, та оформив звіт згідно вимог; при захисті звіту розкриває зміст питання в цілому правильно, але все ж таки студентом допущені помилки, посилання на конкретні історичні періоди та факти, неточності у формулюванні термінів і категорій, проте з допомогою викладача він швидко орієнтується й знаходить правильні відповіді

Практична робота оцінюється на **4-6 балів** коли студент: повністю зробив програму згідно варіанту, та оформив звіт згідно вимог; при захисті звіту відповідь малообґрунтована, неповна; студент лише з допомогою викладача може зрозуміти та виправити свої помилки.

Практична робота оцінюється на **1-3 бал** коли студент: повністю зробив програму згідно варіанту, та оформив звіт згідно вимог; при захисті звіту у відповіді відсутні належні докази й аргументи, зроблені висновки не відповідають загальноприйнятим, хибні; характер відповіді дає підставу стверджувати, що студент не знає правильної відповіді; допущені грубі помилки й студент не може їх виправити.

При здачі та захисту звіту знімається **по 1 балу** коли студент:

* Не оформив звіт відповідно до вимог.
* Недоробив програму згідно завдання чи варіанту.
* Немає UML діаграм чи блок-схем, вони не відповідають реалізованій програмі або генеровані автоматизованими засобами проектування.
* В звіті присутній незрозумілий текст.