線の描画

GDI+

.NET のグラフィックス機能は、Windows の基本的な描画機構である GDI を拡張し
より高度な処理を直感的に行えるようにした GDI+ と呼ばれるもので
これは System.Drawing 名前空間に以下にまとめられています

Form オブジェクトのクライアント領域に図形を描画するには
まず System.Drawing.Graphics オブジェクトを取得しなければなりません

```
Object
   MarshalByRefObject
      Graphics

public sealed class Graphics : MarshalByRefObject, IDisposable
```

このクラスは、描画に関連する様々な基本機能を提供しています
図形や面の塗りつぶし、テキストの描画などには、このオブジェクトが必要です
ただし、このクラスはコンストラクタを非公開としているため、コンストラクタは呼び出せません

一般に、私たちはこのオブジェクトをコントロールから取得しなければいけません
コントロールは System.Windows.Forms.Control クラスで表されます

```
Object
   MarshalByRefObject
      Component
         Control

public class Control : Component, ISynchronizeInvoke, IWin32Window
```

このクラスは、コントロールの基本的な機能を提供するクラスです
ウィンドウもコントロールの一つであり、Form クラスはこのクラスを継承しています

コントロールについては、後ほどより詳しく解説しますが
ここで重要なのは、Form クラスは Control クラスを継承しているということです
そのため、Control クラスが持つ基本的な機能をそのまま使うことができます

Windows プログラムのほとんどは、この Control クラスを知ることにあります
描画を行うには、Control.OnPaint() メソッドをオーバーライドします

protected virtual void OnPaint(PaintEventArgs e);

Control クラスは、プロテクテッドメソッドで、多くの On*() という仮想メソッドを宣言しています
これは イベント を処理するための一つの手段と考えられます
イベントとは、マウスやキーが押されるなどの、ウィンドウに対するユーザーからの要求や
何らかの条件がそろったことをウィンドウに通知するために送られるコールバックを意味します

この OnPaint() メソッドを呼び出すのは、私たちではなく .NET (すなわち システム)です
システムは、ウィンドウが描画処理を必要とした場合に、このメソッドを呼び出します

パラメータの e には、PaintEventArgs オブジェクトが指定されます
このクラスは、描画イベントに必要なデータをカプセル化しています
単純に、描画イベントの発生元から処理先にデータを転送させる手段と考えてください

```
Object
   EventArgs
      PaintEventArgs

public class PaintEventArgs : EventArgs, IDisposable
```

このクラスで最も重要なのは、PaintEventArgs.Graphics プロパティです
この読み取りプロパティが、System.Drawing.Graphics オブジェクトを返してくれます

public Graphics Graphics {get;}

このプロパティから Graphics オブジェクトを受け取れば
後は Graphics クラスの機能を利用して描画処理を行うだけです

まず始めに、単純な線をウィンドウのクライアント領域に描画してみましょう
線の描画には Graphicsc.DrawLine() メソッドを使います

public void DrawLine(Pen pen , int x1 , int y1 , int x2 , int y2);
public void DrawLine(Pen pen , float x1 , float y1 , float x2 , float y2);

pen には、線の描画に用いる Pen オブジェクトを指定します
x1 と y1 には、それぞれ描画の開始位置を表す X 及び Y 座標を
x2 と y2 には、それぞれ線の終点を表す X 及び Y 座標を指定します

.NET におけるペンは System.Drawing.Pen クラスで表されます
ペンとは、枠を描画する時に使われる仮想的なペンを表す GDI オブジェクトの一つで
線の太さや色などの情報を含んでいます

```
Object
   MarshalByRefObject
      Pen
```

```
public sealed class Pen : MarshalByRefObject, ICloneable, IDisposable
```

ペンオブジェクトを作成するには、以下のコンストラクタを使用します
実際には、これ以外にもオーバーロードされたコンストラクタが定義されていますが
Pen クラスについての詳細は「論理ペン」の章で解説します

public Pen(Color color);
public Pen(Color color, float width);

color には、ペンの色を表す Color オブジェクトを指定します
width には、ペンの太さをピクセル単位で指定します

GDI+ では、色を表現するには **System.Drawing.Color** 構造体を用います
この構造体は、システム定義の色や RGB 値などを用いた色の表現を提供します

public struct Color

この構造体は、コンストラクタを公開していません
そのため、直接 new 演算子を用いてインスタンスを作ることはできません
通常は **Color.FromArgb()** 静的メソッドを使って色を生成します

public static Color FromArgb(int argb);
public static Color FromArgb(int red , int green , int blue);
public static Color FromArgb(int alpha , int red , int green , int blue);
public static Color FromArgb(int alpha , Color baseColor);

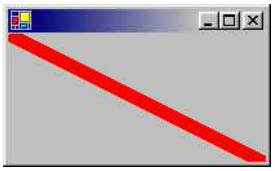
argb には、最上位からバイト単位で、アルファ値、赤、緑、青の色を表す
int 型の 32 ビットカラー値で、一般的には16進数の2桁ずつで表現します
alpha はアルファ、red は赤、green は緑、blue は青の要素を表す 0 ～ 255 までの数値です

alpha と baseColor を指定するシグネチャのメソッドの場合は
baseColor に alpha で指定した新しいアルファ値を採用した色を返します

さて、これで指定した色、指定した幅のペンオブジェクトを作成できます
これを DrawLine() メソッドで指定すれば、そのペンでラインが引かれます
GDI+ のデフォルト座標系は、X 座標は右へ、Y 座標は下へ向かって増加します

```
using System.Windows.Forms;
using System.Drawing;

class WinMain : Form {
 public static void Main(string[] args) {
  WinMain win = new WinMain();
  Application.Run(win);
 }
 override protected void OnPaint(PaintEventArgs e) {
  Graphics g = e.Graphics;
  Pen myPen = new Pen(Color.FromArgb(0xFF , 0 , 0) , 10.0f);
  g.DrawLine(myPen , 0 , 0 , 200 , 100);
 }
}
```



これは、太さ 10 ピクセルの赤いペンで、(0 , 0) から (200 , 100) の位置まで線を引いています
多くのグラフィックス処理は、常にこのような形で行います

スクリーンのある点を示す座標は、2つの数値型で表すこともできますが
それをまとめた **System.Drawing.Point()** 構造体で表すこともできます

public struct Point

この構造体は、主に X と Y 座標を表す情報をパッケージ化する目的を持ち
以下のようなコンストラクタを用いてインスタンスを作成できます

public Point(int dw);
public Point(Size sz);
public Point(int x , int y);

dw には、下位16ビットに X 座標、上位 16 ビットに Y 座標を指定します
sz の場合は、Size オブジェクトの情報を元に座標を決定します
x には X 座標を、y には Y 座標を表す値を直接指定できます

Point 構造体には、座標を表す **Point.X** と **Point.Y** プロパティがあります
このプロパティにアクセスすることによって、オブジェクトの座標を読み書きします

public int X {get; set;}
public int Y {get; set;}

Size オブジェクトとは System.Drawing.Size クラス型のインスタンスで
このクラスは Point 構造体と極めて似た性質を持っていますが
論理的な意味が、座標ではなくサイズ(幅と高さ)を表す点で異なります

public struct Size

この構造体もまた、Point 構造体から作ることも可能になっています
インスタンスの生成には、以下のコンストラクタを用います

public Size(Point pt);
public Size(int width , int height);

pt には、幅と高さを表す Point 構造体のオブジェクトを指定します
X 座標が幅、Y 座標が高さとして扱われます
width には幅を、height には高さを数値で指定します

Size クラスは、幅を表す Size.Width プロパティと
高さを表す Size.Height プロパティを持ちます

public int Width {get; set;}
public int Height {get; set;}

これらは、Size クラスが示す幅と高さにアクセスすることができます
論理的な意味が違うというだけで、実体は Point クラスと同じであることがわかります

実は、Graphics クラスの DrawLine() メソッドは Point 構造体も使えます
これを使えば、カプセル化された座標情報で描画処理を行えるのです

public void DrawLine(Pen pen , Point pt1, Point pt2);
public void DrawLine(Pen pen , PointF pt1 , PointF pt2);

pt1 には始点を、pt2 には終点を表すオブジェクトを指定します
PointF というのは System.Drawing.PointF 構造体のことです
これは、座標の表現に浮動小数を持ちいれるように float 型をメンバに採用したものです
同様に System.Drawing.SizeF 構造体というものも存在します
基本的な機能は Point や Size 構造体と同じなので、詳細は省略します

```
using System.Windows.Forms;
using System.Drawing;

class WinMain : Form {
 Point pt1 , pt2;
 public static void Main(string[] args) {
  WinMain win = new WinMain();
  win.pt1 = new Point(0 , 0);
  win.pt2 = new Point(200 , 100);
  Application.Run(win);
 }
 override protected void OnPaint(PaintEventArgs e) {
  Graphics g = e.Graphics;
  Pen myPen = new Pen(Color.FromArgb(0xFF , 0 , 0) , 10.0f);
  g.DrawLine(myPen , pt1 , pt2);
 }
}
```

このプログラムは、先ほどのプログラムを Point 構造体を用いて再現したものです
結果は同じですが、描画時の座標を Point 構造体で指定しているところが異なります
整数型を直接指定する時よりも、より構造化した美しいプログラムを描けるため
座標が動的に、かつ複雑に変化する場合はこの方法が望まれると思います

連続した線

例えば、DrawLine() メソッドの機能を用いて長方形を描画する場合
少なくとも4回以上もメソッドを呼び出さなければなりません
用いるペンが全て同じ場合でも、この作業が必要となります

このように、同時に同一のペンで複数の線を描画するような場合は
DrawLine() メソッドを何度も呼び出すと、面倒でプログラムの整合性が低下します
そこで、Graphics.DrawLines() メソッドを使うと便利でしょう

public void DrawLines(Pen pen , Point[] points);
public void DrawLines(Pen pen , PointF[] points);

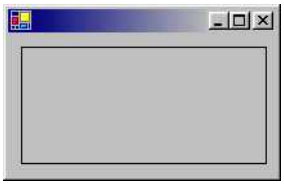
pen にはペンを表す Pen オブジェクトを
points には、描画する座標を表す Point または PointF 構造体の配列を指定します
このメソッドは、指定した Point 配列の先頭から昇順に線を描画していきます

```
using System.Windows.Forms;
using System.Drawing;

class WinMain : Form {
 public static void Main(string[] args) {
  WinMain win = new WinMain();
  Application.Run(win);
 }
 override protected void OnPaint(PaintEventArgs e) {
```

```
  Graphics g = e.Graphics;
  Pen myPen = new Pen(Color.FromArgb(0 , 0 , 0) , 1);
  Point[] pt = {
   new Point(10 , 10) , new Point(200 , 10) ,
   new Point(200 , 100) , new Point(10 , 100) ,
   new Point(10 , 10)
  };

  g.DrawLines(myPen , pt);
 }
}
```



このプログラムでは、DrawLines() に5つの Point 構造体を渡しています
これらは、始点とそれに続く4つの線の座標の位置情報を意味する配列です
複雑なポリゴン処理を行う場合、このような方法で線を引くと便利でしょう

DrawLines() は、単純に連続した線と考えられましたが
ポリゴンを描画したい場合は **Graphics.DrawPolygon()** メソッドが便利です
このメソッドは、DrawLines() に似ていますが、全ての線を描画した後
始点に向けて線を引くことで、自動的に図形を閉じてくれる能力があります

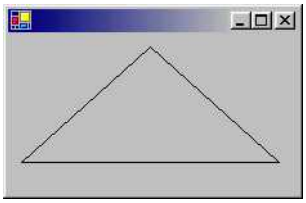
public void DrawPolygon(Pen pen , Point[] points);
public void DrawPolygon(Pen pen , PointF[] points);

pen には、線を引く時のペンを表す Pen オブジェクトを
points には、ポリゴンの各頂点の位置を表す Point または PointF 構造体を指定します

```
using System.Windows.Forms;
using System.Drawing;

class WinMain : Form {
 public static void Main(string[] args) {
  WinMain win = new WinMain();
  Application.Run(win);
 }
 override protected void OnPaint(PaintEventArgs e) {
  Graphics g = e.Graphics;
  Pen myPen = new Pen(Color.FromArgb(0 , 0 , 0) , 1);
  Point[] pt = {
   new Point(110 , 10) , new Point(210 , 100) ,
   new Point(10 , 100)
  };

  g.DrawPolygon(myPen , pt);
 }
}
```



このプログラムは DrawPolygon() メソッドを用いて三角形を描画しています
DrawLine() の論理的な意味は、線の視点とそれに続く終点の位置の配列でしたが
DrawPolygon() の配列は、ポリゴンの頂点の座標を表しています

前のページへ　　戻る　　次のページへ