論理ブラシ

図を塗りつぶす

これまでの閉じた図形は、単純に輪郭をペンで描画しただけのものでした
では、長方形や円弧を特定の色などで塗りつぶしたい場合はどうするのでしょうか

Graphics クラスでは、Draw* というメソッドの他に、似たようなメソッドで
Fill* という塗りつぶし専用のメソッド郡を提供しています
図形の内部を塗りつぶしたい場合は、この Fill* という種類のメソッドを使います

長方形を塗りつぶすには **Graphics.FillRectangle()** メソッドを使います

public void FillRectangle(Brush brush , Rectangle rect);
public void FillRectangle(Brush brush , RectangleF rect);
public void FillRectangle(Brush brush , int x , int y , int width , int height);
public void FillRectangle(Brush brush , float x , float y , float width , float height);

brush には、塗りつぶしに用いる論理ブラシを示す Brush オブジェクトを
rect には、長方形の領域を表す構造体のオブジェクトを指定します
x と y は、長方形の左上角を示す座標、width と height は幅と高さを指定します

第一パラメータを除けば、DrawRectangle() メソッドと同じです
塗りつぶしの描画では、論理ペンの代わりに**論理ブラシ**を用います
これは **System.Drawing.Brush** 抽象クラスで表されます

```
Object
   MarshalByRefObject
      Brush

public abstract class Brush : MarshalByRefObject, ICloneable, IDisposable
```

なぜ、ブラシはペンと異なり抽象クラスで表現されているのでしょうか？
それは、ブラシという性質上、単純に色で塗りつぶすだけではなく
必要に応じて、画像を使ったテクスチャや、複雑な図形で塗りつぶしてもよいと考えられ
ブラシはこの要求に応えられるように、Brush を拡張して用途に応じて使い分けます

最も簡単なブラシは **System.Drawing.SolidBrush** クラスです
このブラシは、単一の色で塗りつぶす時の用いることができます

public sealed class SolidBrush : Brush

このように、SolidBrush クラスは Brush 抽象クラスを継承しています
こうして、Brush 型に暗黙的変換を行うことができるように設計されているのです
このクラスの公開コンストラクタは次のように定義されています

public SolidBrush(Color color);

color には、ブラシの色を示す Color オブジェクトを指定します
作成した SolidBrush の色へは **SolidBrush.Color** プロパティでアクセスできます

public Color Color {get; set;}

必要に応じて、このプロパティから色を取得したり変更することができます
単純な1色の塗りつぶしには、このブラシが有効です

```
using System.Windows.Forms;
using System.Drawing;
using System.Drawing.Drawing2D;

class WinMain : Form {
 public static void Main(string[] args) {
  Application.Run(new WinMain());
 }
 override protected void OnPaint(PaintEventArgs e) {
  Graphics g = e.Graphics;
  Brush myBrush = new SolidBrush(Color.FromArgb(0 , 0xFF , 0));

  g.FillRectangle(myBrush , new Rectangle(10 , 10 , 200 , 100));
 }
}
```

このプログラムは、緑色の SolidBrush によって長方形が塗りつぶされています
もし、複数の色を混在させたグラデーションで塗りつぶしたいならば
System.Drawing.Drawing2D.LinearGradientBrush クラスを使います
System.Drawing.Drawing2D 名前空間は、高度な2次元グラフィック処理を提供しています

public sealed class LinearGradientBrush : Brush

これは、簡単な設定だけで美しいグラデーションを描画できる強力なブラシです
コンストラクタは次のようなものが定義されています

```
public LinearGradientBrush(
 Point point1 , Point point2 ,
 Color color1 , Color color2
);
public LinearGradientBrush(
 PointF point1 , PointF point2 ,
 Color color1 , Color color2
);
public LinearGradientBrush(
 Rectangle rect , Color color1 , Color color2 ,
 LinearGradientMode linearGradientMode
);
public LinearGradientBrush(
 Rectangle rect ,
 Color color1 , Color color2 , float angle
);
public LinearGradientBrush(
 RectangleF rect , Color color1 , Color color2 ,
 LinearGradientMode linearGradientMode
);
public LinearGradientBrush(
 RectangleF rect ,
 Color color1 , Color color2 , float angle
);
public LinearGradientBrush(
 Rectangle rect , Color color1 , Color color2 ,
 float angle , bool isAngleScaleable
);
public LinearGradientBrush(
 RectangleF rect , Color color1 , Color color2 ,
 float angle , bool isAngleScaleable
);
```

point1 にはグラデーションの開始地点、point2 は終端地点を示す構造体を指定します
color1 はグラデーションの開始色、color2 は終端の色を指定します
塗りつぶし処理では、指定した範囲で color1 から color2 に向けて色が変化します

rect の場合は、グラデーションで塗りつぶす範囲を構造体で指定します
linearGradientMode には、どの方向で塗りつぶすかを指定する列挙型です
この列挙型については、すぐ後で詳しく解説します

angle は塗りつぶし範囲を表す矩形の X 座標を基点に
塗りつぶす方向の角度を時計回りで指定します
isAngleScaleable が true であれば、このクラスと結び付けられている変換処理に
angle が影響を受けるということを表し、そうでなければ false を指定します

LinearGradientBrush クラスは、ジオメトリ変換処理の設定用メソッドを提供していて
isAngleScaleable はこの変換処理が角度に影響するかどうかを表すものですが
この場では、ジオメトリ変換などは扱わないため省略します

```
using System.Windows.Forms;
using System.Drawing;
using System.Drawing.Drawing2D;

class WinMain : Form {
 public static void Main(string[] args) {
  Application.Run(new WinMain());
 }
 override protected void OnPaint(PaintEventArgs e) {
  Graphics g = e.Graphics;
  Rectangle rect = new Rectangle(10 , 10 , 200 , 100);
  Brush myBrush = new LinearGradientBrush(
   rect , Color.FromArgb(0xFF , 0 , 0) ,
   Color.FromArgb(0 , 0 , 0xFF) , 45.0f
  );

  g.FillRectangle(myBrush , rect);
 }
}
```

図のように、プログラムを実行すると綺麗なグラデーションの長方形が描画されます
LinearGradientBrush ブラシを使えば、このようなグラデーションを簡単に実現できます
JPEG に圧縮しているためグラデーションが粗いですが、実際はもっと美しいです

長方形を指定するコンストラクタのケースで、グラデーションの方向を指定する場合
System.Drawing.Drawing2D.LinearGradientMode 列挙型を使う方法もあります

public enum LinearGradientMode

この列挙型のメンバは、次のものが定義されています

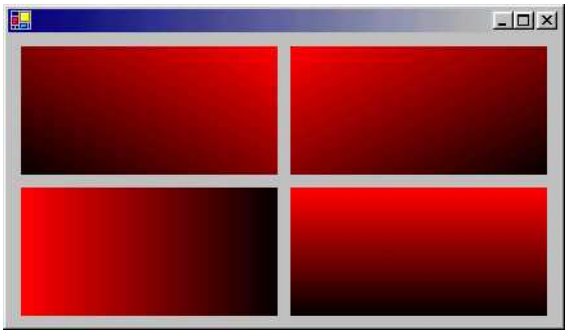
定数	解説
BackwardDiagonal	右上から左下に勾配を指定
ForwardDiagonal	左上から右下に勾配を指定
Horizontal	左から右に勾配を指定
Vertical	上から下に勾配を指定

この列挙型のメンバを用いて、方向を指定することができます

```
using System.Windows.Forms;
using System.Drawing;
using System.Drawing.Drawing2D;

class WinMain : Form {
 public static void Main(string[] args) {
  Application.Run(new WinMain());
 }
 override protected void OnPaint(PaintEventArgs e) {
  Graphics g = e.Graphics;
  Rectangle rect = new Rectangle(10 , 10 , 200 , 100);
  LinearGradientMode[] mode = {
   LinearGradientMode.BackwardDiagonal ,
   LinearGradientMode.ForwardDiagonal ,
   LinearGradientMode.Horizontal ,
   LinearGradientMode.Vertical
  };
  for (int i = 0 ; i < 4 ; i++) {
   Brush myBrush = new LinearGradientBrush(
    rect , Color.FromArgb(0xFF , 0 , 0) ,
    Color.FromArgb(0 , 0 , 0) , mode[i]
   );

   g.FillRectangle(myBrush , rect);
   if (i == 1) { rect.X = 10; rect.Y += 110; }
   else rect.X += 210;
  }
 }
}
```



このプログラムは、それぞれの列挙型メンバの効果を確認するためのものです
グラデーションの色は、赤から黒に向かいます

また、単色に対して単純な網目模様を用いた効果を用いたい場合
System.Drawing.Drawing2D.HatchBrush クラスを用いると良いでしょう

public sealed class HatchBrush : Brush

このクラスのコンストラクタは、次のように定義されています

public HatchBrush(HatchStyle hatchstyle , Color foreColor);
public HatchBrush(HatchStyle hatchstyle , Color foreColor , Color backColor);

hatchstyle には、ブラシの模様を定めるスタイルを指定します
foreColor は模様ラインの色を、backColor には背景色を指定します

ハッチブラシの模様は、第一引数のハッチスタイルで決定されます
これは、System.Drawing.Drawing2D.HatchStyle 列挙型を用います

public enum HatchStyle

この列挙型では、以下のメンバが定義されています
模様は、文章では説明が難しいのでそれぞれ実験して試してください

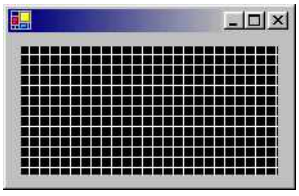
メンバ	
BackwardDiagonal	Cross
DarkDownwardDiagonal	DarkHorizontal
DarkUpwardDiagonal	DarkVertical

DashedDownwardDiagonal	DashedHorizontal
DashedUpwardDiagonal	DashedVertical
DiagonalBrick	DiagonalCross
Divot	DottedDiamond
DottedGrid	ForwardDiagonal
Horizontal	HorizontalBrick
LargeCheckerBoard	LargeConfetti
LargeGrid	LightDownwardDiagonal
LightHorizontal	LightUpwardDiagonal
LightVertical	Max
Min	NarrowHorizontal
NarrowVertical	OutlinedDiamond
Percent05	Percent10
Percent20	Percent25
Percent30	Percent40
Percent50	Percent60
Percent70	Percent75
Percent80	Percent90
Plaid	Shingle
SmallCheckerBoard	SmallConfetti
SmallGrid	SolidDiamond
Sphere	Trellis
Vertical	Wave
Weave	WideDownwardDiagonal
WideUpwardDiagonal	ZigZag

```
using System.Windows.Forms;
using System.Drawing;
using System.Drawing.Drawing2D;

class WinMain : Form {
 public static void Main(string[] args) {
  Application.Run(new WinMain());
 }
 override protected void OnPaint(PaintEventArgs e) {
  Graphics g = e.Graphics;
  Rectangle rect = new Rectangle(10 , 10 , 200 , 100);
  Brush myBrush = new HatchBrush(
   HatchStyle.Cross ,  Color.FromArgb(0xFF , 0xFF , 0xFF));

  g.FillRectangle(myBrush , rect);
 }
}
```



このプログラムでは Cross スタイルのハッチブラシで矩形を塗りつぶしています
この時、前景色（線の色）は白に設定し、背景色はデフォルトにしてあります

ハッチブラシの情報は **HatchBrush.BackgroundColor**
HatchBrush.ForegroundColor 及び
HatchBrush.HatchStyle プロパティで取得することができます

public Color BackgroundColor {get;}
public Color ForegroundColor {get;}
public HatchStyle HatchStyle {get;}

BackgroundColor はブラシの背景色、ForegroundColor は前景色
HatchStyle はハッチスタイルを返す読み取り専用プロパティです
ハッチブラシオブジェクトの情報が必要になった場合に用いると良いでしょう

さて、この他にも画像を使ったブラシなども存在しますが
画像などについては、まだ説明していないので後ほど紹介します

前のページへ　　戻る　　次のページへ