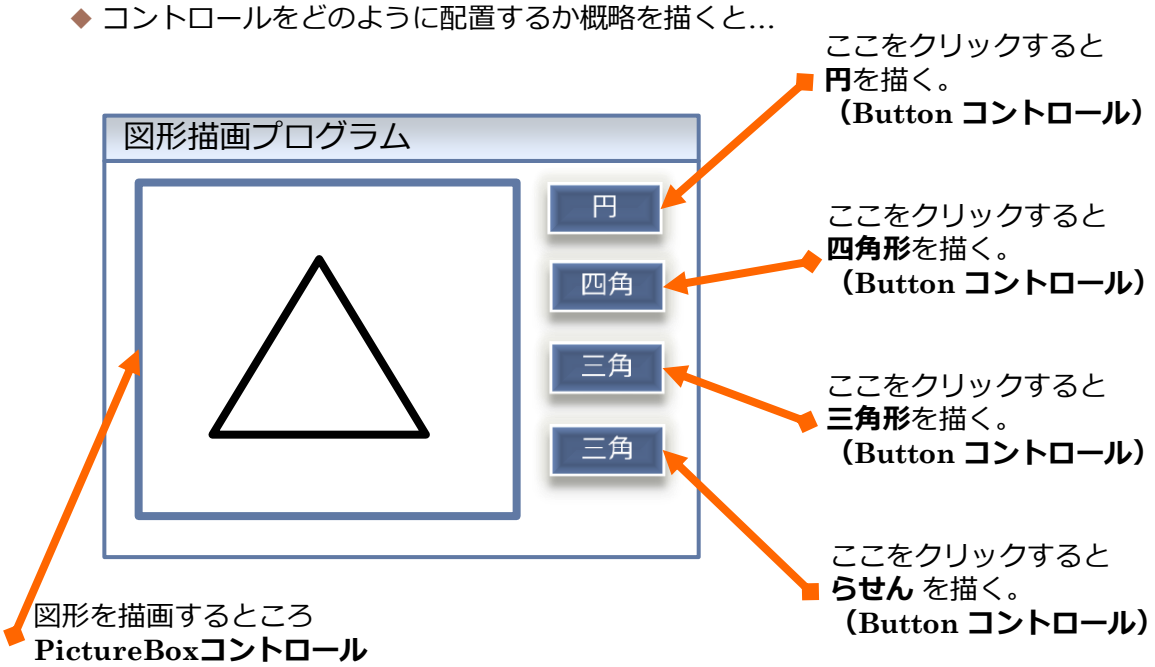
【02】 グラフィックで簡単な図形を描く
図形描画プログラム

1 今回作成するアプリケーションの概要

ボタンをクリックすると図形を描くプログラム

- 行われる動作
  - [1] ボタンをクリック
  - [2] そのボタンに対する図形を描く
- これを使用者とコンピュータの関係で描くと
  - [使用者 → コンピュータ] ボタンをクリック
  - [使用者 ← コンピュータ] 図形を描画して見せる。
- 使用者がコンピュータにすること
  - ボタンをクリック→ **Button コントロール**
- コンピュータがすること
  - 図形を描く
- コンピュータが使用者にすること
  - 描いた図形を示したい！ → **PictureBox コントロール**
- 必要なコントロールは次の通り
  - Button コントロール（コンピュータにどの図形を描くかを示す）
  - PicuterBox コントロール（BMIを表示する）
- コントロールをどのように配置するか概略を描くと…



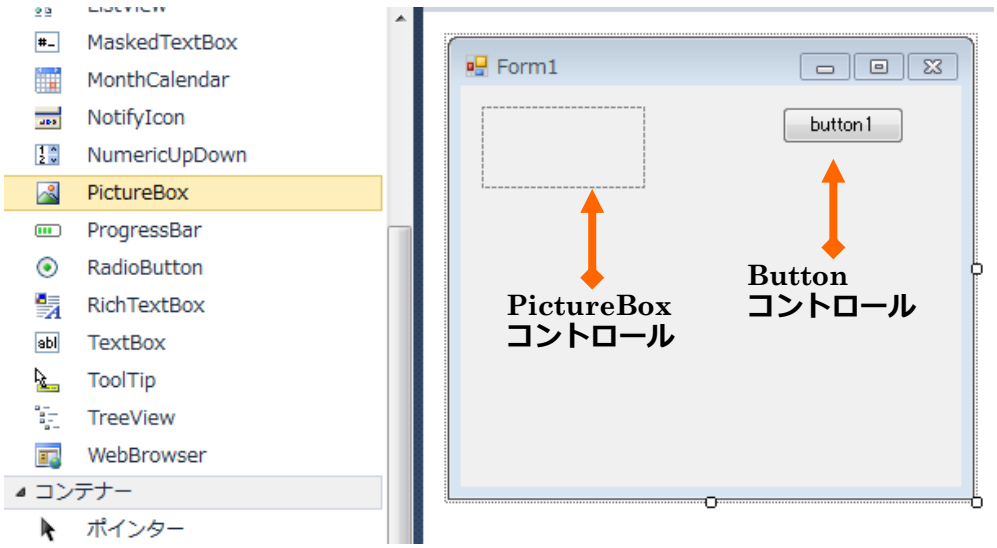
2 Visual Studio 2010 の起動と新規プロジェクトの作成

- 前回やったとおり、Visual Studio 2010 を起動
  [1]「スタート」→ [2]「すべてのプログラム」→ [3]「Visual Studio 2010」のフォルダ
  → [4]「Visual Studio 2010」のアイコン

- 新規プロジェクトも前回の手順で作成
  [1] メニュー →「ファイル」→ [2]「新規作成」→ [3]「プロジェクト」
  [4]「Visual C#」→ [5]「Windowsフォームアプリケーション」
  [6]「プロジェクト名」を入力 **（今回は JKJ02）**
  [7]「参照…」をクリックして、プロジェクトを保存する場所を選択
  [8]「ソリューションのディレクトリを作成」のチェックは はずす
  [9]「OK」をクリック

3 コントロールの配置

- PictureBox コントロール 1 個、Button コントロール 1 個を
  ツールボックスからドラッグ&ドロップして配置
- 場所は後で調節するので適当に配置
- ボタンは 1 個だけ配置。残りのボタンはあとで配置する。

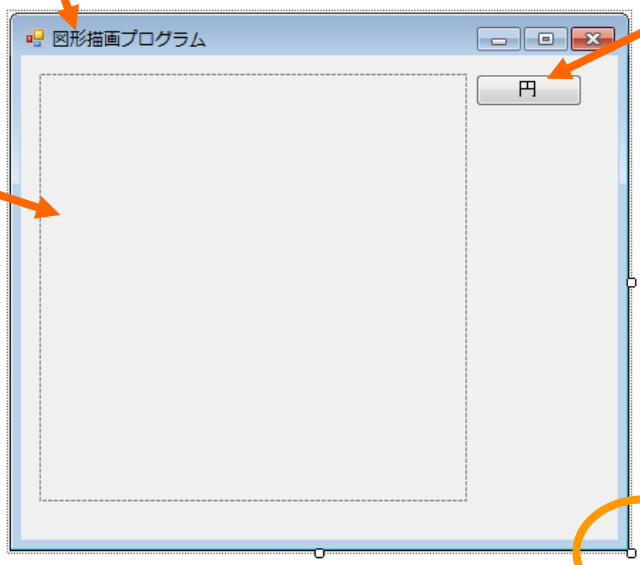


■下の図の通り、コントロールのプロパティを設定し、配置を調整する

Name	Form1
Text	図形描画プログラム

Name	btnCircle
Text	円

Name	pbCampus
Size	300, 300

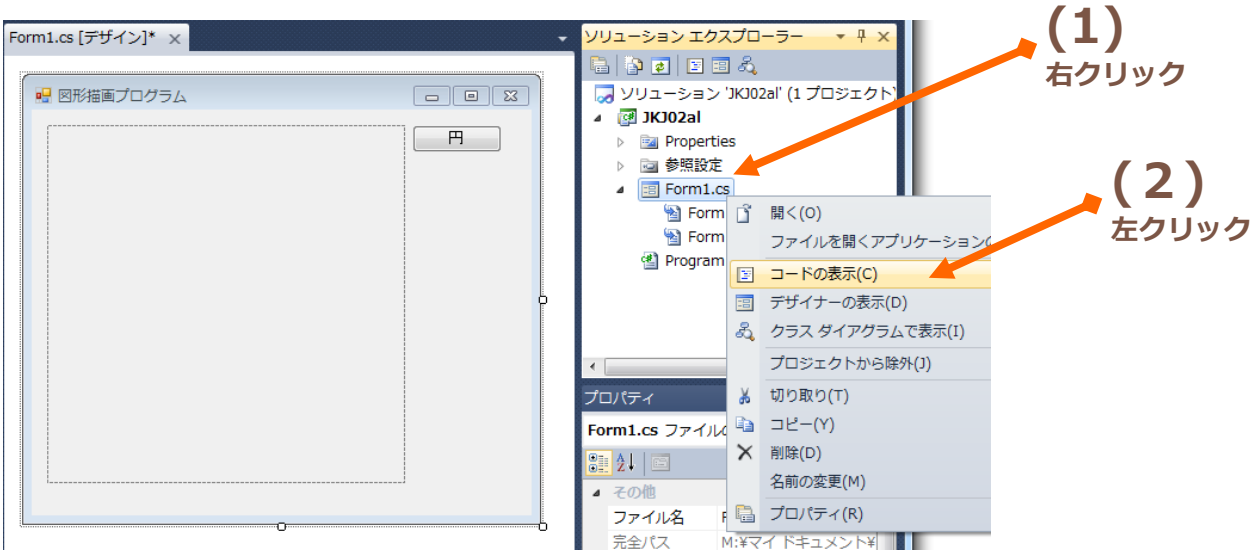


位置はマウスでドラッグして、だいたいこのあたりに配置する

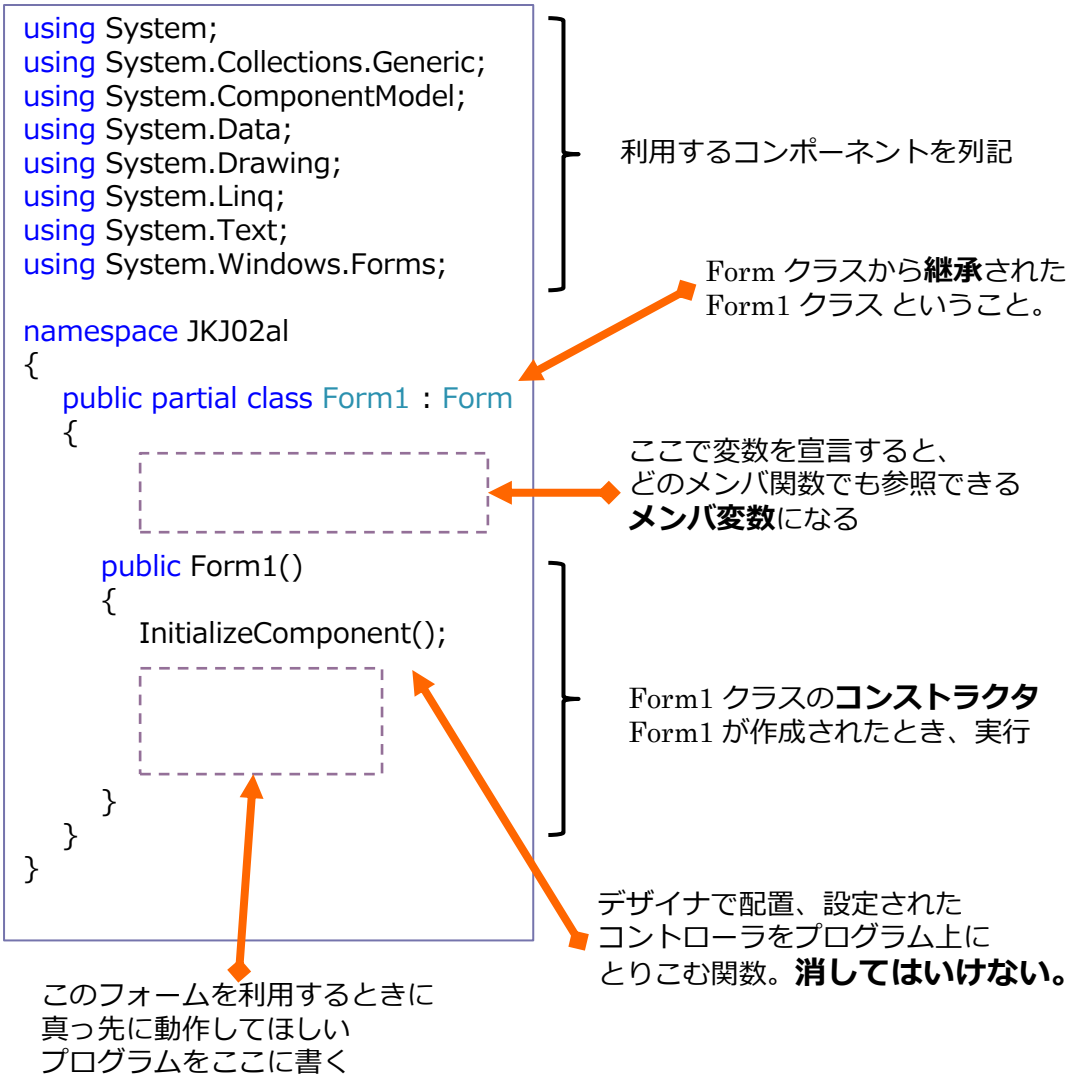
このあたりをドラッグすると、フォームの大きさを変えることができる。
だいたいこれぐらいの大きさにする

4 グラフィック描画の準備のプログラムの入力

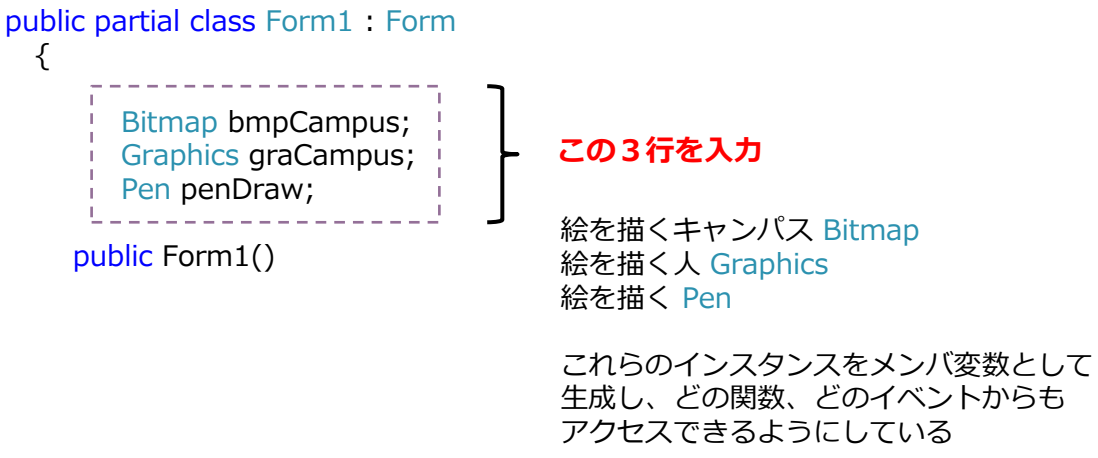
(1) ソリューションエクスプローラー の Form.cs で、右クリック
(2) コードの表示をクリック



(3) Form.cs のプログラムが表示される



(4) メンバ変数を宣言する部分に、以下のプログラムを入力する



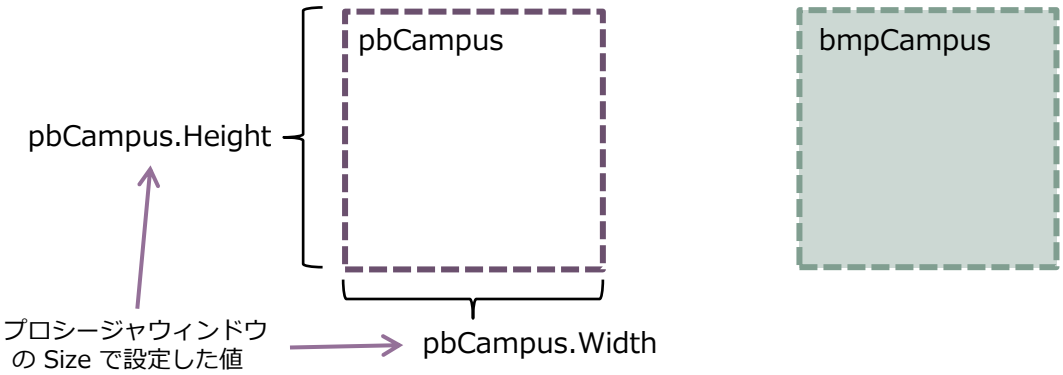
(5) コンストラクタの部分に以下の４行を入力

```
public Form1()
 {
        InitializeComponent();
        bmpCampus = new Bitmap(pbCampus.Width, pbCampus.Height);
        pbCampus.Image = bmpCampus;
        graCampus = Graphics.FromImage(pbCampus.Image);
        penDraw = new Pen(Color.Black, 3);
 }
```

この４行を入力

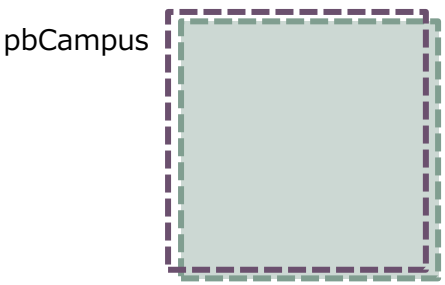
```
bmpCampus = new Bitmap(pbCampus.Width, pbCampus.Height);
```

pbCampus と同じ大きさのビットマップを作成する



```
pbCampus.Image = bmpCampus;
```

pbCampus の Image に bmpCampus を関連付ける



こうすれば、ウィンドウの再描画の処理を
pbCampus に任せることができる

```
graCampus = Graphics.FromImage(pbCampus.Image);
```

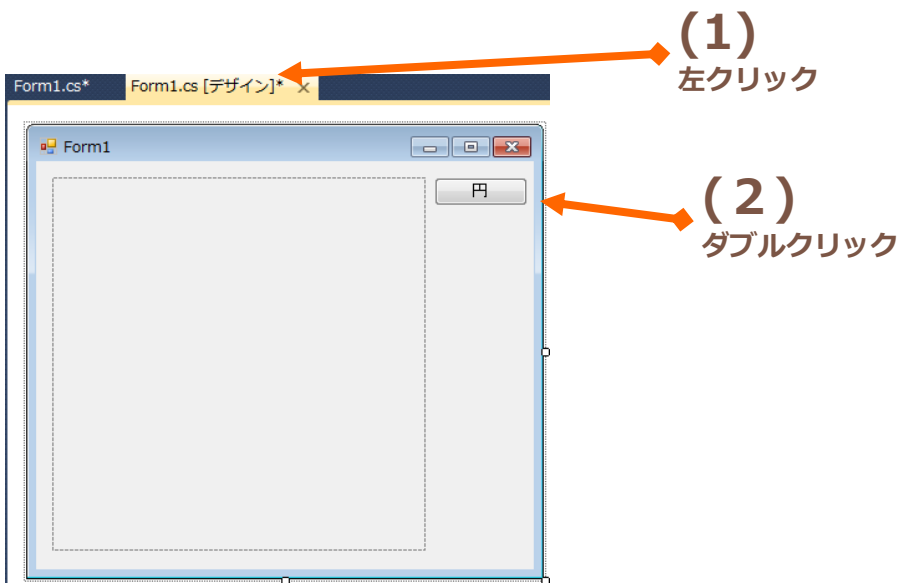
graCampus は pbCampmus の Image に描くようにする。

```
penDraw = new Pen(Color.Black, 3);
```

ペンの設定を「黒色」で「太さ3」にする

5 「円」ボタンを押されたときのプログラムを入力

(1) From1.cs [デザイン] タグをクリックして、フォームのデザインするタグへ戻る
(2) 「円」ボタン（btnCircle）をダブルクリック



(3) プログラムを入力するタグに移動し、自動的に btnCircle_Click 関数が作られる。
この btnCircle_Click 関数の中身を次のように入力する。

```
private void btnCircle_Click(object sender, EventArgs e)
{
    graCampus.Clear(Color.White);
    graCampus.DrawEllipse(penDraw, 50, 50, 200, 200);
    pbCampus.Refresh();
}
```

この３行を入力

```
graCampus.Clear(Color.White);
```

graCampus は Graphics.FromImage(pbCampus.Image) で作成されているので、これのメソッド（メンバ関数）を呼び出すことは、pbCampus にグラフィックを描くことになる。

Graphics の メソッド Clear は指定された色でグラフィックを消去する。

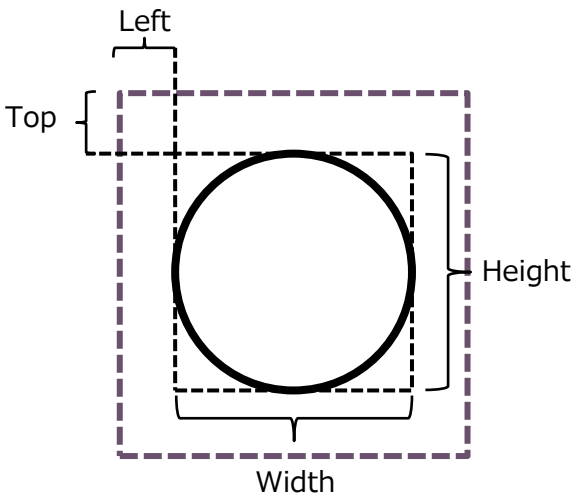
すなわち、graCampus.Clear(Color.White); は pbCampus を　白色で塗りつぶすということを行う。

```
graCampus.DrawEllipse(penDraw, 50, 50, 200, 200);
```

DrawEllipse メソッドは楕円を描く。楕円の縦方向と横方向の長さが等しければ円になる。

描くときの線の色や太さは Pen クラスで示す。
今回はすでに生成しておいた penDraw （黒色で太さ 3） を使う。

50, 50 は楕円を囲む四角形の左上の座標 (Left, Top)。
200, 200 は楕円を囲む四角形の幅 Width と高さ Height。

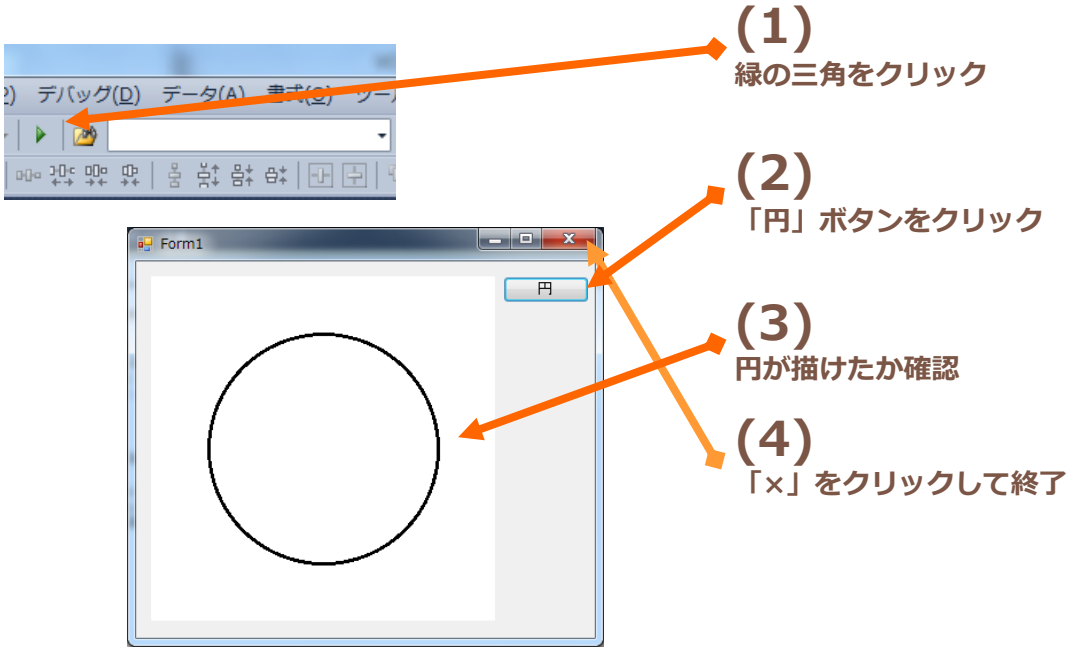



```
pbCampus.Refresh();
```

描かれたグラフィックスは適当なタイミングを見計らって表示される。
自分で意図的に描画結果を表示させたいときは Refresh() を呼び出す必要がある。

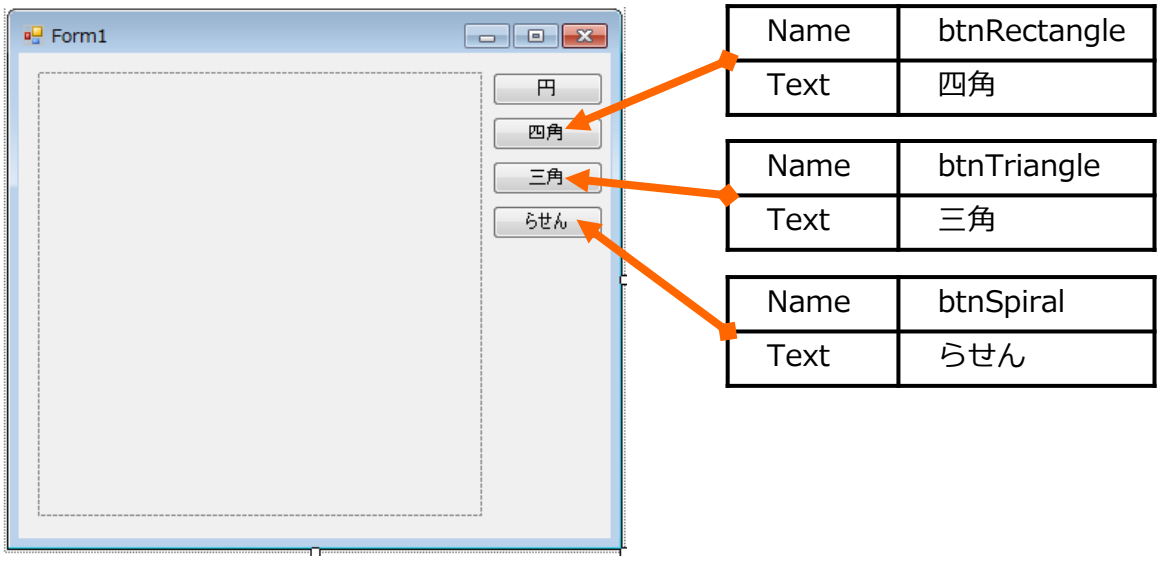
6 「円」ボタンを押したときの動作確認

(1) 緑の三角をクリックしてプログラムをコンパイルして、起動
(2) 「円」ボタン（btnCircle）をクリック
(3) 円が描画されるのを確認
(4) 「×」をクリックしてプログラムを止める



7 「四角」「三角」「らせん」ボタンの配置

Button コントロールを 3 つ図のように配置し、プロパティを設定する。



Name	btnRectangle
Text	四角

Name	btnTriangle
Text	三角

Name	btnSpiral
Text	らせん

8 「四角」ボタンを押されたときのプログラムを入力

「四角」ボタンをダブルクリックして、btnRectangle_Click 関数を自動的に生成して、以下のプログラムを入力する。

```
private void btnRectangle_Click(object sender, EventArgs e)
    {

        graCampus.Clear(Color.White);
        graCampus.DrawRectangle(penDraw, 50, 50, 200, 200);
        pbCampus.Refresh();

    }
```

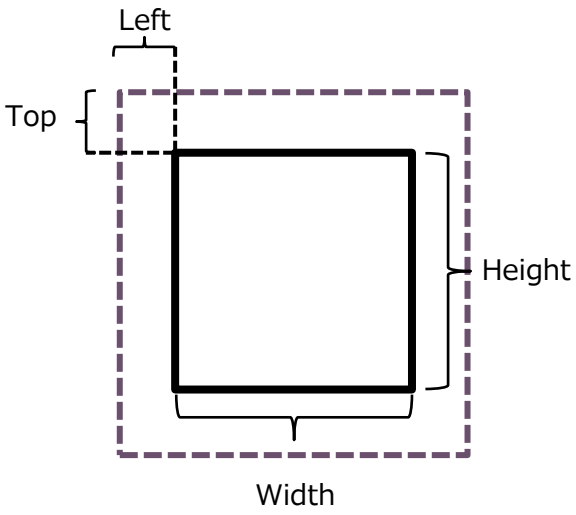
この３行を入力

```
graCampus.DrawRectangle(penDraw, 50, 50, 200, 200);
```

DrawRectangle メソッドは四角形を描く。縦と横の長さが等しければ正方形になる。

描くときの線の色や太さは Pen クラスで示す。
今回はすでに生成しておいた penDraw （黒色で太さ 3） を使う。

50, 50 は四角形の左上の座標 (Left, Top)。
200, 200 は四角形の幅 Width と高さ Height。



9 「三角」ボタンを押されたときのプログラムを入力

「三角」ボタンをダブルクリックして、btnTriangle_Click 関数を自動的に生成して、
以下のプログラムを入力する。

三角形を描く関数はないので、３本の線を引いて三角形を描いている。

```
private void btnTriangle_Click(object sender, EventArgs e)
    {
        graCampus.Clear(Color.White);
        graCampus.DrawLine(penDraw, 150, 50, 50, 250);
        graCampus.DrawLine(penDraw, 150, 50,250, 250);
        graCampus.DrawLine(penDraw,  50,250,250, 250);
        pbCampus.Refresh();
    }
```

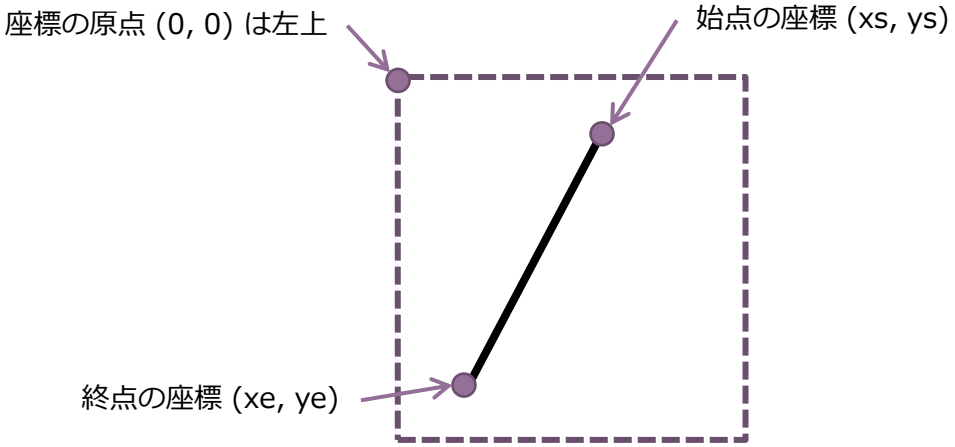
この５行を入力

```
graCampus.DrawLine(penDraw, 150, 50, 50, 250);
```

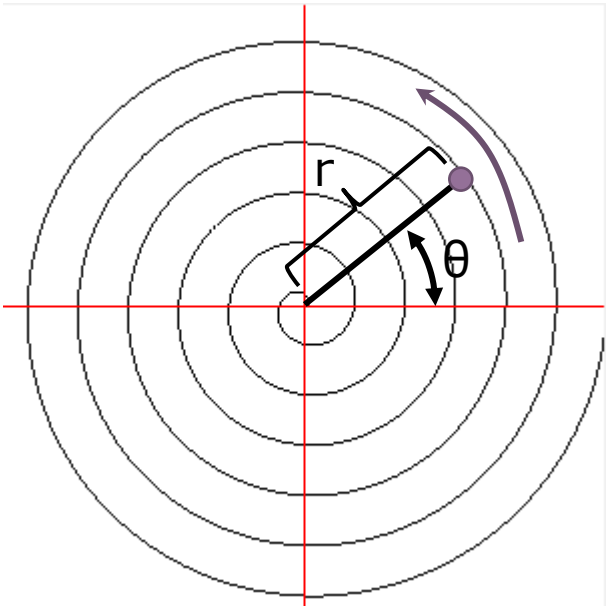
DrawRectangle メソッドは線を描く。

描くときの線の色や太さは Pen クラスで示す。
今回はすでに生成しておいた penDraw （黒色で太さ 3） を使う。

150,50 は線の始点の座標 (xs, ys) 、
50, 250 は線の終点の座標 (xe, ye) である。



１０ 「らせん」とは



左の図のように、原点から遠ざかりながら離れていく点の軌跡を描いたものが「らせん」である。

回転角度をθ、原点と回転移動する点との距離を r とすると、

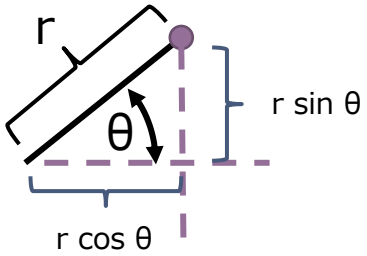
$$r = a\theta$$

になる。ここで、 a は定数である。

原点からの距離 r と角度 θ で
表されている点の座標は極座標形式

これをグラフィックで表現するためには
直交座標系形式に変換する必要がある

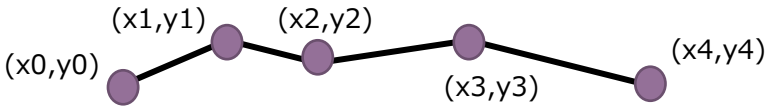
$$x = r\cos\theta$$
$$y = r\sin\theta$$



１１ 曲線を描くアルゴリズム

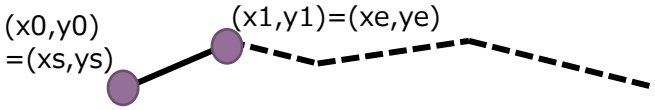
らせんの曲線を描くために、多くの曲線上の点を計算する。
この多くの点を短い直線で結んで、曲線を表現する。

(0) まず、計算される点と描かれる曲線が次のようだとする。

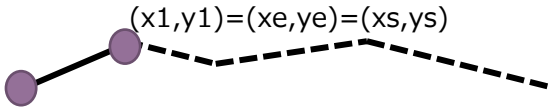


(1) 最初の点 (x0,y0) を計算し、(xs, ys) の座標へ格納

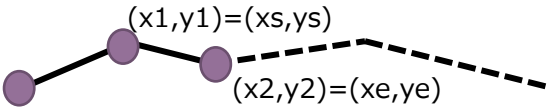
(2) 次の点（x1,y1) を計算し、(xe, ye) の座標へ格納。
(xs,ys) と (xe, ye) の間で直線を描く



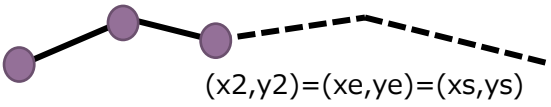
(3) (xs, ys) を (xe, ye) で置き換える。



(4) 次の点（x2,y2) を計算し、(xe, ye) の座標へ格納。
(xs,ys) と (xe, ye) の間で直線を描く



(5) (xs, ys) を (xe, ye) で置き換える。



(6) これを繰り返して、曲線を描く

１２　「らせん」ボタンを押されたときのプログラムを入力

「三角」ボタンをダブルクリックして、btnSpiral_Click 関数を自動的に生成して、
以下のプログラムを入力する。

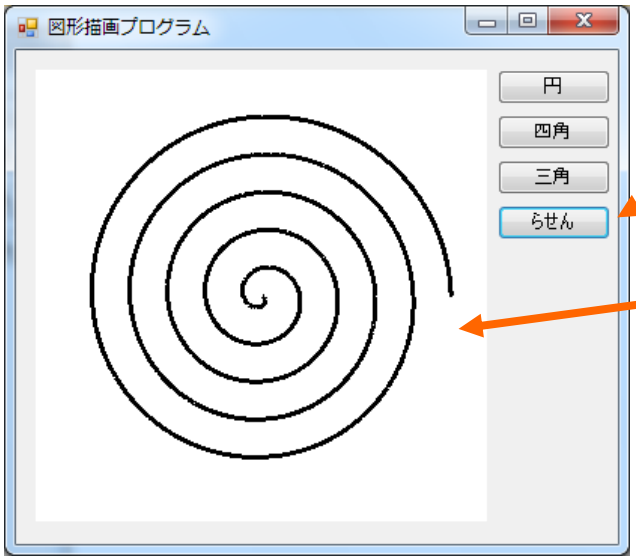
```
private void btnSpiral_Click(object sender, EventArgs e)
    {
        graCampus.Clear(Color.White);
        double r, xs, ys, xe, ye;
        double a = 4;
        xs = 150.0;
        ys = 150.0;
        for (double theta = 0.0; theta < 10.0 * Math.PI; theta += 0.02 * Math.PI)
        {
            r = a * theta;
            xe = r * Math.Cos(theta) + 150.0;
            ye = r * Math.Sin(theta) + 150.0;
            graCampus.DrawLine(penDraw, (float)xs, (float)ys, (float)xe, (float)ye);

            xs = xe;
            ys = ye;
        }
        pbCampus.Refresh();

    }
```

この部分を入力

Math.PI　：円周率
Math.Cos(theta)　：theta ラジアン のときの cos
Math.Sin(theta)　：theta ラジアン のときの sin

※ 数学関連の関数や定数は Math で始まる。

１３ 「らせん」ボタンを押したときの実行結果



(1)
「らせん」ボタンを クリック

(3)
らせんが描けたか 確認

１４ まとめ

- PictureBox にグラフィックを描くために
  - [1] Bitmap を用意する
  - [2] 用意したBitmap を PictureBox の Image に関連付ける
  - [3] Graphics のインスタンスを PictureBox の Image により生成
  - [4] Graphics のメソッド（メンバ関数）を利用してグラフィックを描く

- フォームのメンバ変数とコンストラクタのプログラムを入力するために
  - [1]ソリューションエクスプローラー の Form.cs で、右クリック
  - [2]「コード」の表示をクリック

- 今回使ったコントロール
  - Button コントロール（クリックすることで何かイベントを生じさせる）
  - PictureBox コントロール（画像を表示したり、描画を管理したり）NEW！

- 今回使った Graphics のメソッド（メンバ関数）
  - DrawEllipse ：楕円を描く
  - DrawRectangle ：四角形を描く
  - DrawLine ：線を描く
  - Clear ：ぶらふぃっくを消去する

- 今回使った算術関係の定数と関数
  - Math.PI ： 円周率
  - Math.Cos(theta) ： theta ラジアン のときの cos
  - Math.Sin(theta) ： theta ラジアン のときの sin

１５ 追加課題

1 TextBox を追加して、そこの値によって図形の大きさを指定できるようにする。

2 ウィンドウの大きさを変更したら、それに合わせて pictureBox, Button の場所や大きさが変化するようにする。 pictureBox の大きさが変更したら、合わせてグラフィックの表示も変更になるように。

3 「五角形」のボタンを追加する。このボタンが押されたら正五角形を描くようにプログラムも追加する。