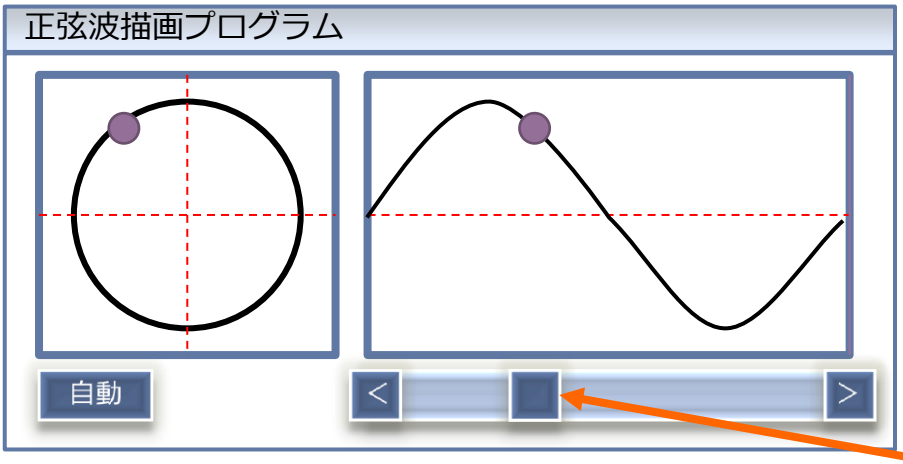
【03】 簡単なアニメーションを実現してみる
正弦波描画プログラム

1 今回作成するアプリケーションの概要

正弦波が円周上の点の動きから描かれることを表すプログラム

◆ 行われる動作
[1] 起動すると円と正弦波が描かれる
[2] マウスで移動するスライダを動かすと円周上の点と正弦波上の点が連動して動く
[3] ボタンをクリックすると、連動している二つの点がそれぞれ円周上と正弦波上を自動的に移動する



このバーをマウスでドラッグして左右に動かすことができる（スライダ）

◆ 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] 円と正弦波を描画して見せる。

[使用者 → コンピュータ] スライダを移動
[使用者 ← コンピュータ] 円と正弦波を描画し、同じ位相の場所に点を描く。

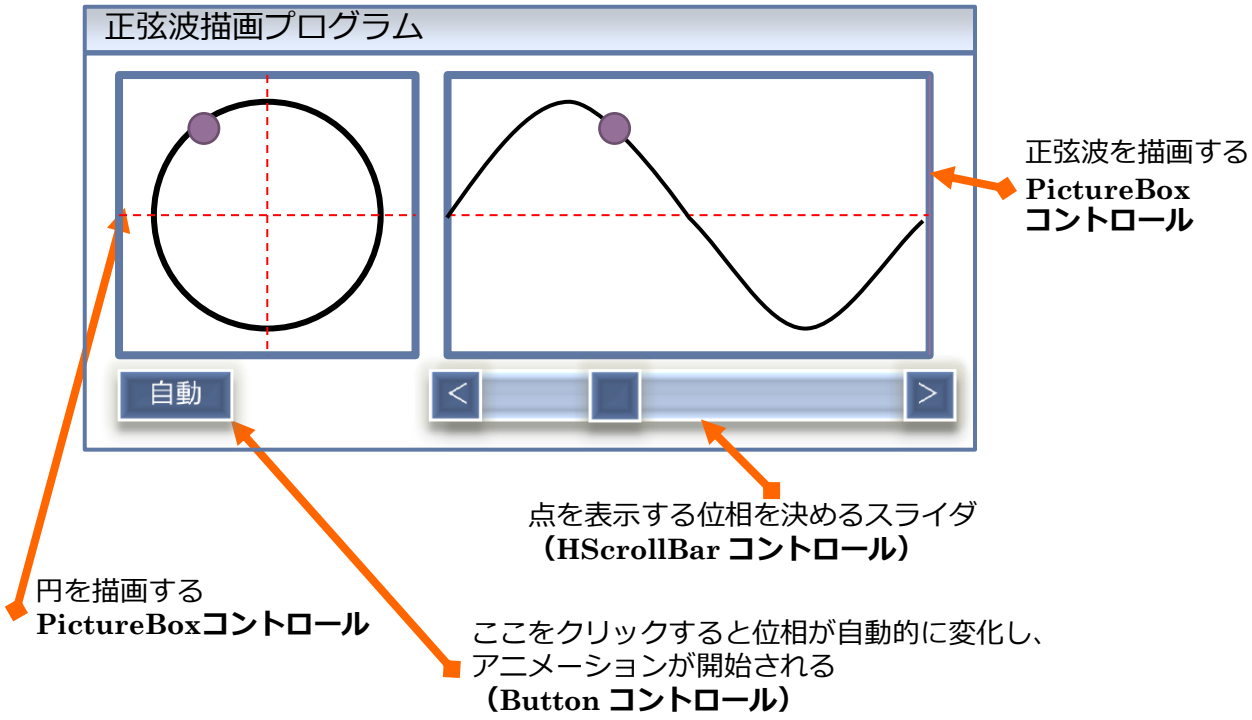
[使用者 → コンピュータ] 「自動」ボタンをクリック
[使用者 ← コンピュータ] 円と正弦波を描画し、連動して円周上と正弦波上の点を移動させる。
「自動」ボタンの表示を「停止」に変更

[使用者 → コンピュータ] 「停止」ボタンをクリック
[使用者 ← コンピュータ] 円と正弦波を描画し、連動して円周上と正弦波上の点を停止させる。
「停止」ボタンの表示を「自動」に変更

◆ 自動的に動かすためには
関数を一定の時間間隔で呼び出し、その関数で処理。
→ この処理を行うのが **Timer 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（「自動」と「停止」を行う）
- ◆ PicuterBox コントロール（円 と 正弦波 を表示する）
- ◆ HScrollBar コントロール（ドラッグで値を変更するスライダ）
- ◆ Timer コントロール（一定時間間隔で処理、アニメーションのために必要、）

◆ コントロール配置の概略図



■ **Timer コントロール** はフォーム上に配置されない

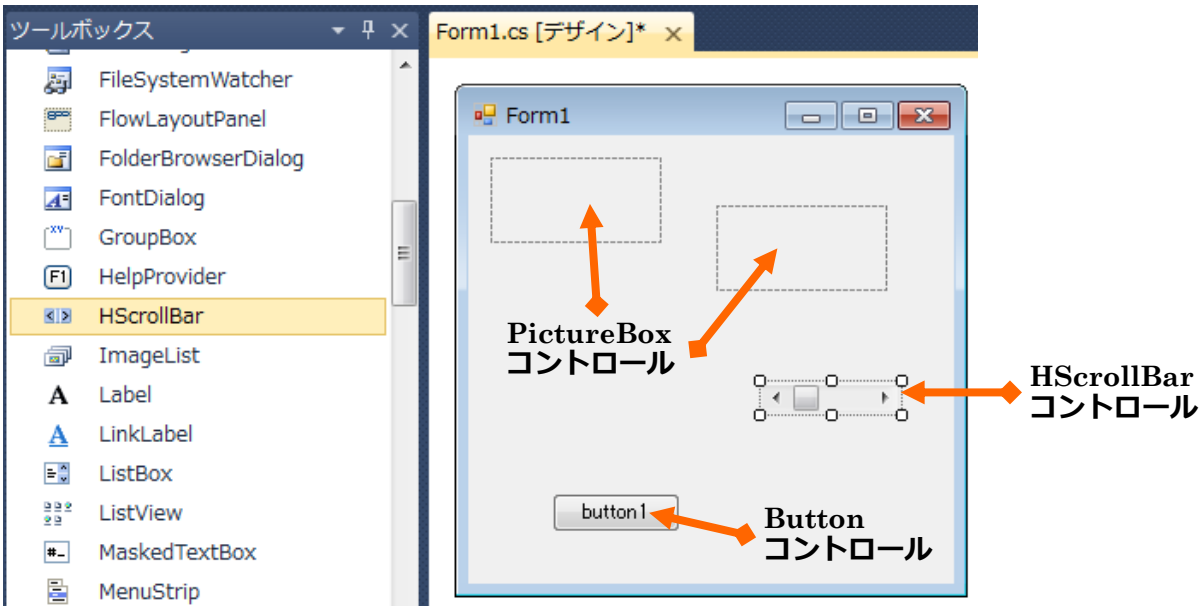
2 Visual Studio 2010 の起動と新規プロジェクトの作成

- 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

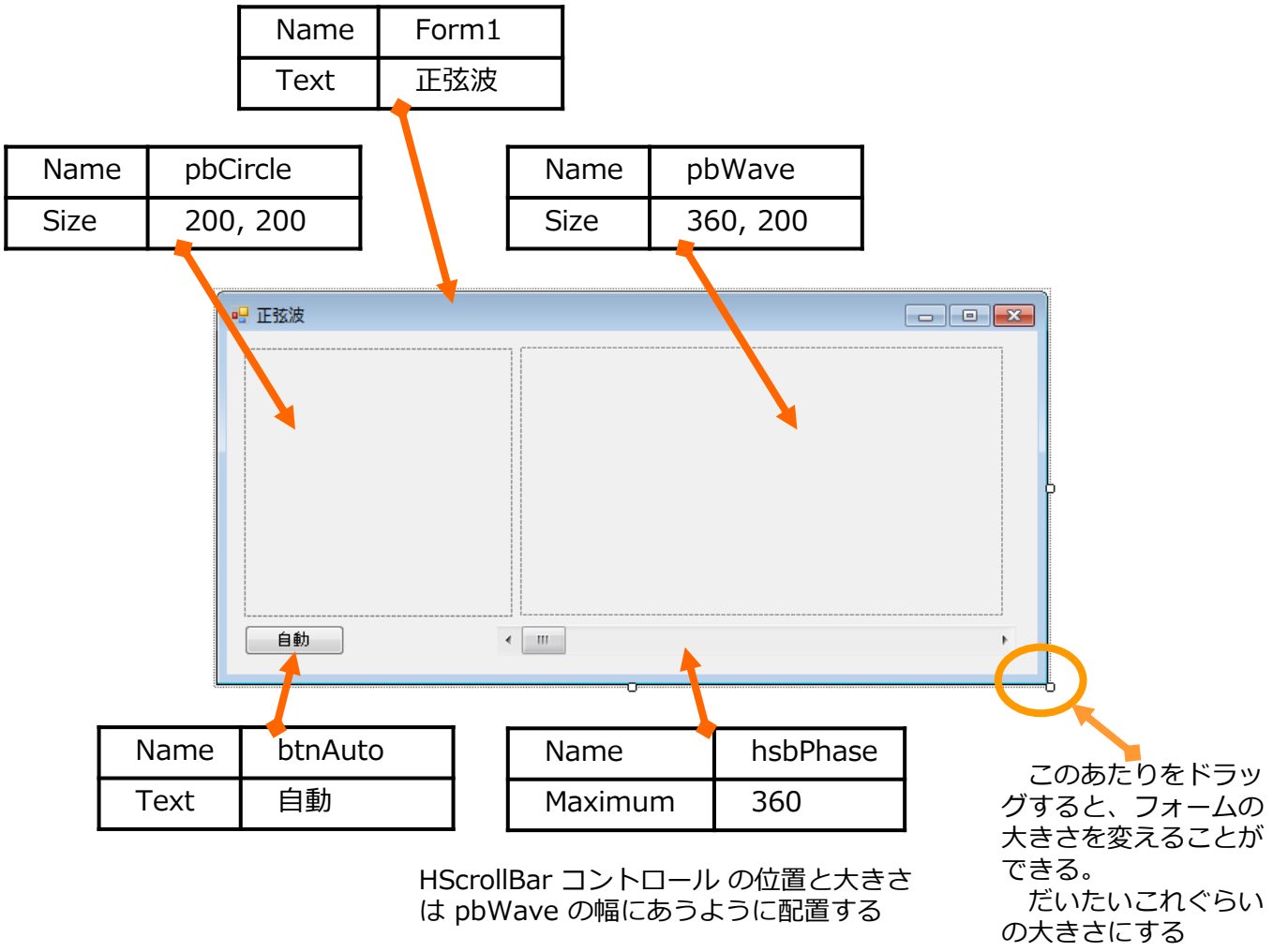
- 新規プロジェクトも前回の手順で作成
  [1] メニュー →「ファイル」→ [2]「新規作成」→ [3]「プロジェクト」
  [4]「Visual C#」→ [5]「Windowsフォームアプリケーション」
  [6]「プロジェクト名」を入力 **（今回は JKJ03）**
  [7]「参照…」をクリックして、プロジェクトを保存する場所を選択
  [8]「ソリューションのディレクトリを作成」のチェックは はずす
  [9]「OK」をクリック

3 コントロールの配置

- PictureBox コントロール 2 個、Button コントロール 1 個、
  HScrollBar 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 Size プロパティを設定してからで調節するので、適当に配置
- Timerコントロールは後で配置
  （円と正弦波が表示され、スライダで連動できるようになってから）

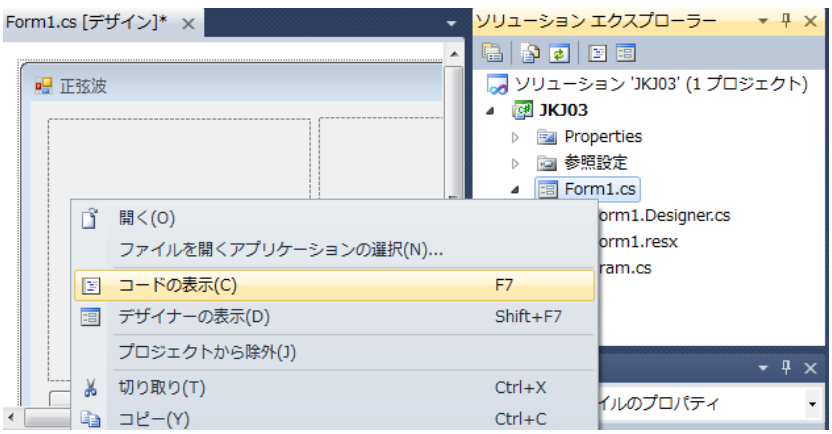


■下の図の通り、コントロールのプロパティを設定し、配置を調整する



4 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
今回は PictureBox コントロールが2つあるので、2つづつ入力

```
public partial class Form1 : Form
{
    Bitmap bmpCircle;
    Graphics graCircle;

    Bitmap bmpWave;
    Graphics graWave;

    Pen penAxis;      // 座標軸を描くペン
    Pen penWave;      // 円、波形を描くペン
    Pen penNow;       // スクロールバーが示す位相の場所を表す円を描くペン

    double Radius;  // 円の半径 = 正弦波の振幅
    int Theta;      // スクロールバーが示す値と連動させる

    public Form1()
    {
```

この部分を入力

(5) コンストラクタの部分に以下の部分を入力

```
public Form1()
    {
        InitializeComponent();

        bmpCircle = new Bitmap(pbCircle.Width, pbCircle.Height);
        pbCircle.Image = bmpCircle;
        graCircle = Graphics.FromImage(pbCircle.Image);

        bmpWave = new Bitmap(pbWave.Width, pbWave.Height);
        pbWave.Image = bmpWave;
        graWave = Graphics.FromImage(pbWave.Image);

        penAxis = new Pen(Color.Black, 1);
        penAxis.DashStyle = System.Drawing.Drawing2D.DashStyle.Dash;
            // 軸は 点線 で引きたい
        penWave = new Pen(Color.Black, 1);
        penNow = new Pen(Color.Blue, 2);

        Theta = 0;
        Radius = 80;

    }
```

この部分を入力

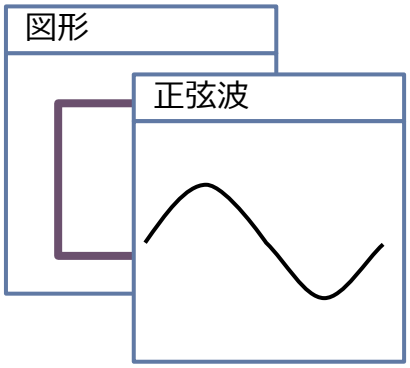
5 フォームの再描画イベントの作成

再描画イベントとは

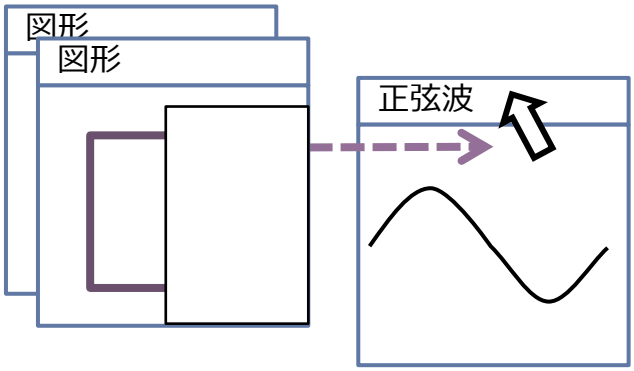
重なって背面にいたウィンドウが表へ出てきたり、最小化されていたウィンドウが再び表示されたりして、隠されてたウィンドウが再び描画されても、表示がされてなかったり、欠けたりしないよう、この再描画が必要なときにイベントを発生させている。

この時生じるイベントを再描画イベントといい、これはウィンドウの重なりや最小化から表示されたり、ウィンドウの大きさが変更されたりしたきに呼び出される。

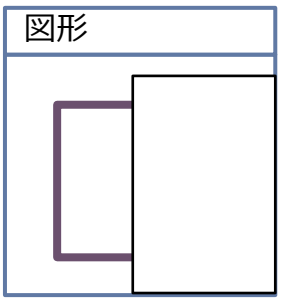
再描画イベント時に呼び出される関数でグラフィックを書いておくと、ウィンドウの変化してもそのグラフィック表示を維持してくれる。



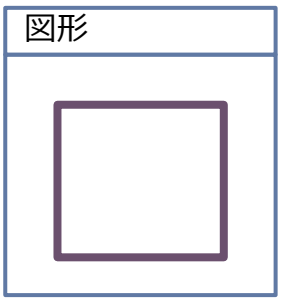
1. ウィンドウが重なっている



2. 上のウインドウを動かす。
下のウィンドウの重なっている部分には何も表示されない（先の表示が消される）

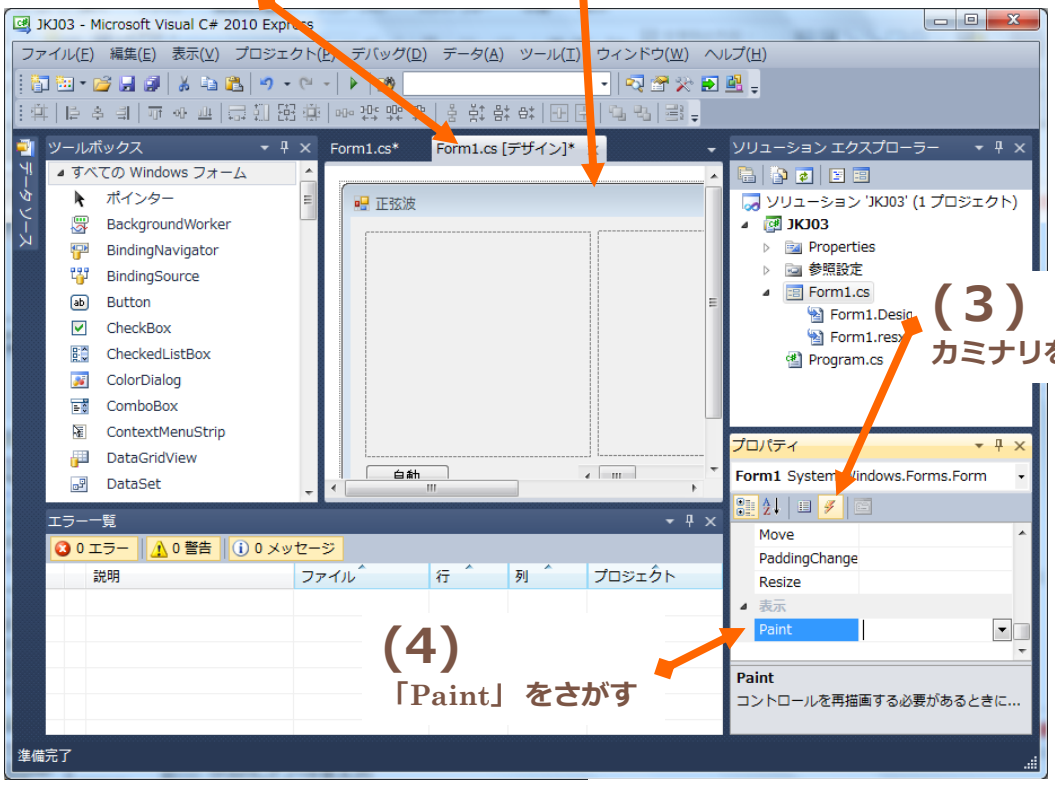


3. 再描画イベントが発生



4. 再描画イベントでウィンドウ全部の描画を行うようにしてある。だから、重なって表示されなかった分も表示

(1) Form1.cs [デザイン] をクリックして、コード入力のタグからフォームのデザインするタグへ戻る
(2) From1 のプロパティを選ぶ。（フォームのコントロールの配置してない場所をクリック）
(3) プロパティウィンドウのカミナリのところをクリックしてイベント一覧を表示
(4) イベント一覧の中から「Paint」を探す
(5) Paint の右側のテーブルをクリックしてカーソルが点滅した状態にする
(6) [Enter] キーを押す



(5)「Paint」 の右側のセルをクリック

(6)「Paint」 の右側のセルでカーソルが点滅していたら、[Enter]キーを押す

(7) Form1_Paint という関数が作られる。
この Form1_paint が再描画のときに実行される関数である。

```
private void Form1_Paint(object sender, PaintEventArgs e)
    {

    }
```

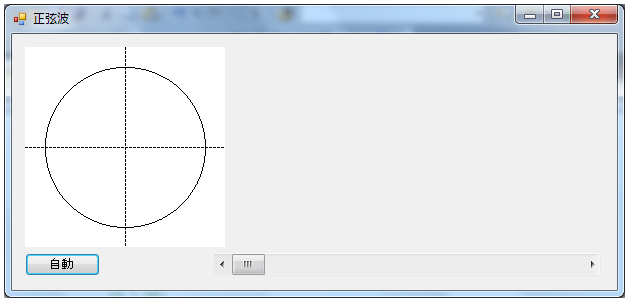
6 フォームの再描画時のプログラムの入力

- 今回は再描画時にほとんどのグラフィック描画を実施する

```
private void Form1_Paint(object sender, PaintEventArgs e)
    {
        double xs, ys;  // 線の始点
        double xe, ye;  // 線の終点
        double xz, yz;  // 座標の原点の PictureBox 上の座標

        // 円を描くほうのピクチャボックス pbCampus から描く
        // 描画領域を白色でクリア
        graCircle.Clear(Color.White);

        // pbCircle の中心の座標を求める
        xz = pbCircle.Width / 2;
        yz = pbCircle.Height / 2;

        // 座標軸を描く
        graCircle.DrawLine(penAxis, (float)xz, 0, (float)xz, pbCircle.Height);
        graCircle.DrawLine(penAxis, 0, (float)yz, pbCircle.Width, (float)yz);

        // 極座標形式で円を描く
        xs = xz + Radius * Math.Cos(0);
        ys = yz - Radius * Math.Sin(0);

        for (int th = 0; th < 360; th++)
        {
            double rth = Math.PI * th / 180.0;
            xe = xz + Radius * Math.Cos(rth);
            ye = yz - Radius * Math.Sin(rth);

            graCircle.DrawLine(penWave, (float)xs, (float)ys, (float)xe, (float)ye);

            xs = xe;
            ys = ye;
        }
}
```

この部分を入力

この段階で実行すると、右図のように、円だけ描かれる



■ 続けて次の部分を入力する

```
            xs = xe;
            ys = ye;
        }
```

前のページで入力した続きから

```
        // アニメーションする点を描く
        xs = xz + Radius * Math.Cos(Theta / 180.0 * Math.PI);
        ys = yz - Radius * Math.Sin(Theta / 180.0 * Math.PI);
        graCircle.DrawEllipse(penNow, (float)(xs - 4), (float)(ys - 4), 9, 9);
```

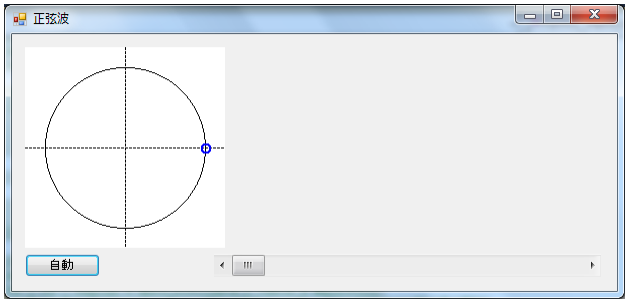
この部分を入力

```
    }
```

これは位相 Theta の点を描いてる。
Theta は 初期値は 0 だから、横軸上に青い丸が表示されている

スクロールバー や タイマーでこの Theta を変更することで青い丸を移動させる

この段階で実行すると、右図のように、円周上に青い丸が描かれる



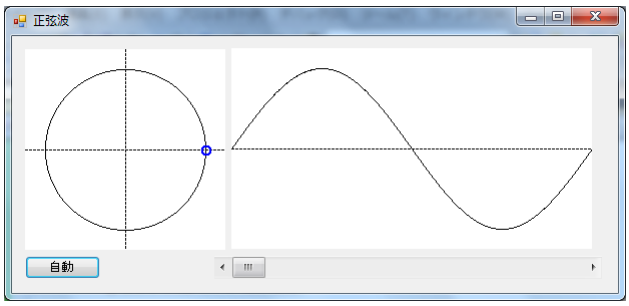
■ 続けて次の部分を入力する、今度は正弦波を表示する部分である。

前のページで入力した続きから

```
// 正弦波を描くほうのピクチャボックス pbWave を描く
// 描画領域を白色でクリア
graWave.Clear(Color.White);

// グラフの原点は右端、縦方向は真ん中
xz = 0;
yz = pbWave.Height / 2;

// 縦軸 = 0になる線を引く
graWave.DrawLine(penAxis, 0, (float)yz, pbWave.Width, (float)yz);

// 正弦波を描く
xs = 0;
ys = yz - Radius * Math.Sin(0);

for (int th = 0; th < 360; th++)
{
    double rth = Math.PI * th / 180.0;
    xe = xz + th / 360.0 * pbWave.Width;
    ye = yz - Radius * Math.Sin(rth);

    graWave.DrawLine(penWave, (float)xs, (float)ys, (float)xe, (float)ye);

    xs = xe;
    ys = ye;
}
```

この部分を入力

この段階で実行すると、右図のように、正弦波も表示する



■ 続けて次の部分を入力する。正弦波上に Theta の位相の点を表示する部分である。

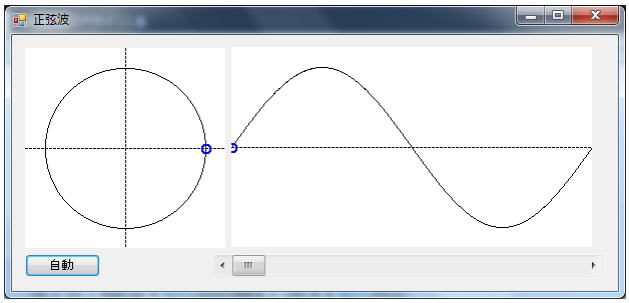
前のページで入力した続きから

```
        // アニメーションする点を描く
        xs = xz + Theta / 360.0 * pbWave.Width;
        ys = yz - Radius * Math.Sin(Theta / 180.0 * Math.PI);
        graWave.DrawEllipse(penNow, (float)(xs - 4), (float)(ys - 4), 9, 9);
```

この部分を入力

```
}
```

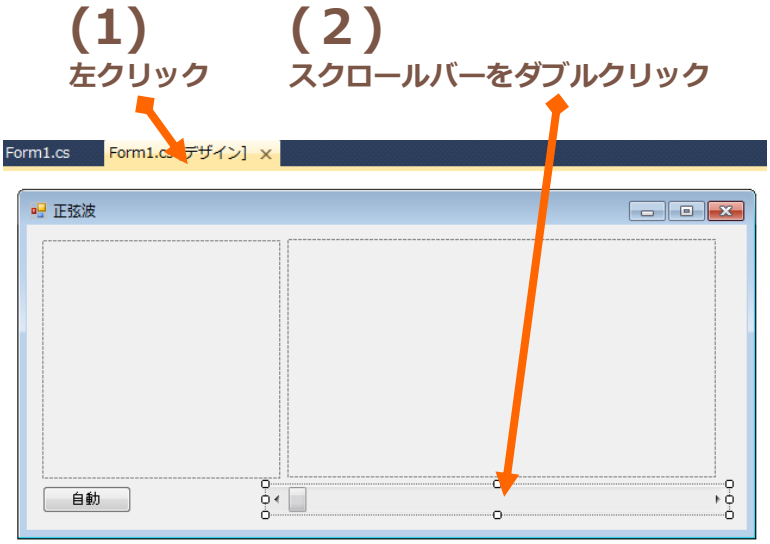
この段階で実行すると、右図のように、正弦波上にも青い丸が描かれる



7 スクロールバー変更時のプログラムの入力

スクロールバーをマウスでスライドさせるとスクロールバーの持つ値が変更になる。
このスクロールバーが変更されたイベントのとき、呼び出される関数を次の手順で生成する。

(1) Form1.cs [デザイン] をクリックして、コード入力のタグからフォームのデザインするタグへ戻る
(2) スクロールバーをダブルクリック



(3) 関数 hsbPhase_Scroll が生成されるので、次の 2 行を入力する

```
private void hsbPhase_Scroll(object sender, ScrollEventArgs e)
{
    Theta = hsbPhase.Value;
    Refresh();
}
```

この２行を入力

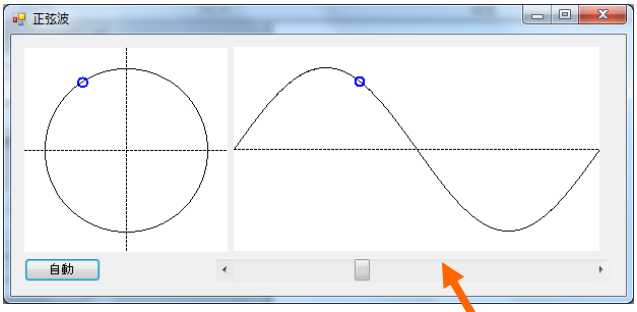
Theta = hsbPhase.Value;

hsbPhse のスクロールのある位置から得られる値を Theta へ

Refresh( );

フォームの再描画イベントを強制的に呼び出す

(4) コントロールの配置とプログラムの入力が行われたら、ためしに動作させてみよう

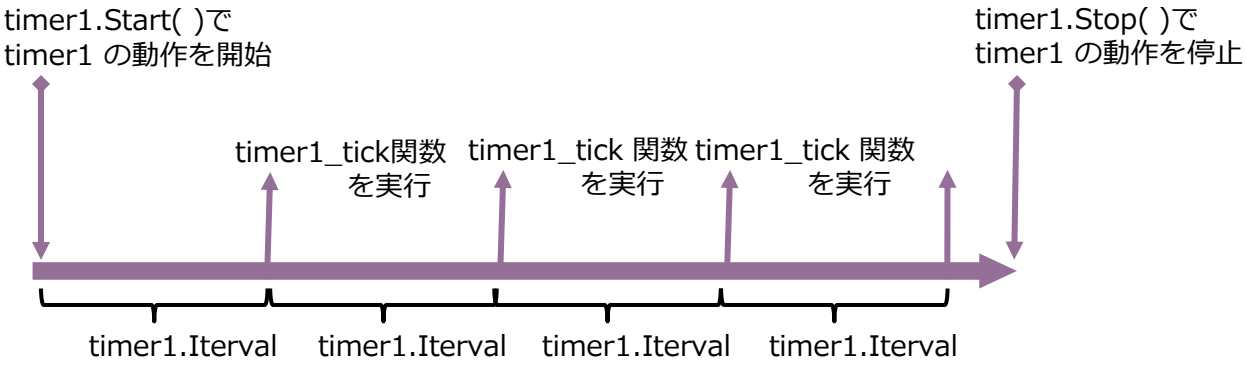


水平スクロールバーをスライドすると円周と正弦波の上の ○ が移動する

8 タイマーの配置

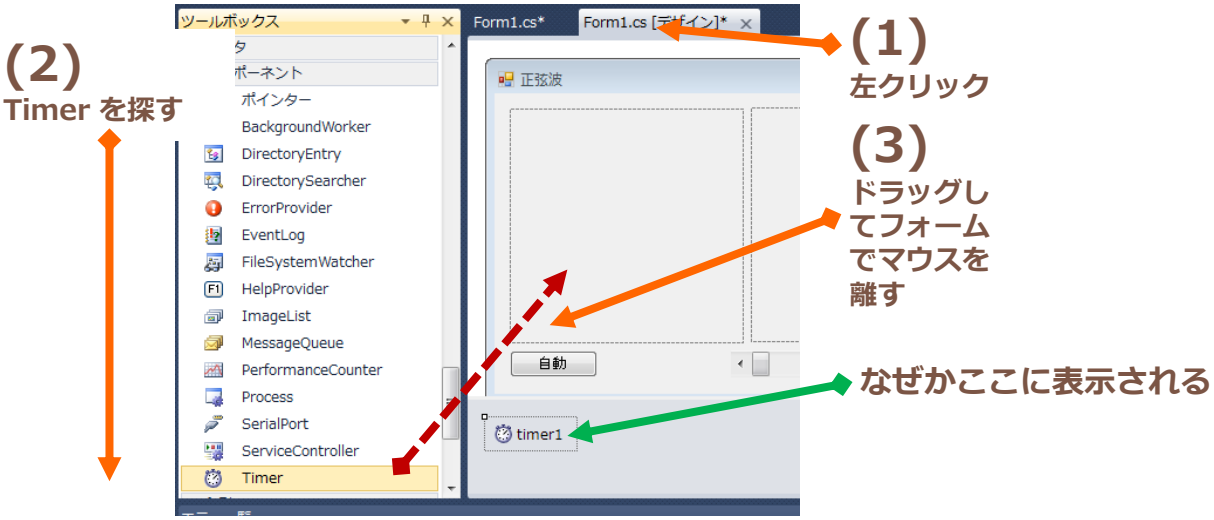
「自動」ボタンをクリックすると青い○ が自動的に円と円周上を移動するように作りたい。

このような場合には、ある一定の時間ごとに特定の関数を呼び出す Timer コントロールを利用する。



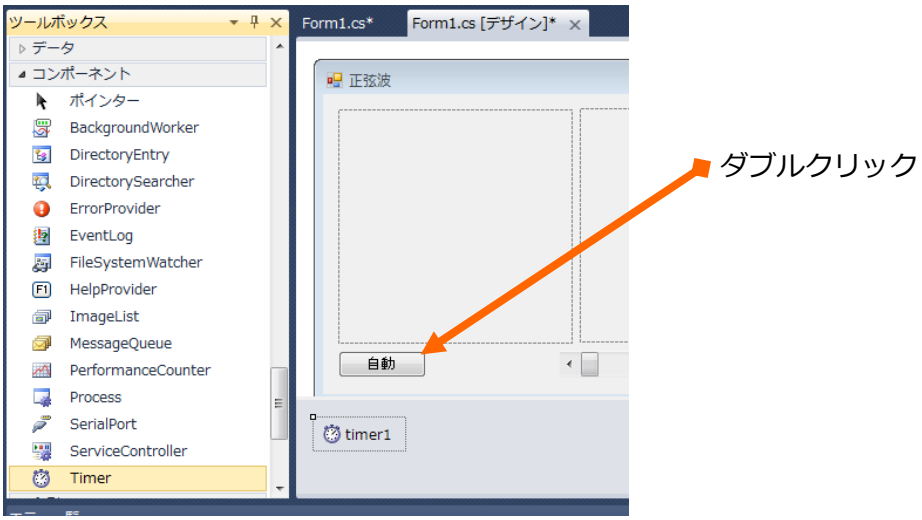
Timer コントロール timer1 の動作

(1) Form1.cs [デザイン] をクリックして、コード入力のタグからフォームのデザインするタグへ戻る
(2) ツールボックスから Timer コントロールを探す
(3) ツールボックスの Timer をドラッグして、フォームの設計画面上で離す。



9 タイマーの操作

「自動」ボタンをクリックすると
　　タイマーが動作を開始する
　　ボタンの表示を「停止」に変更
「停止」ボタンをクリックすると
　　タイマーが動作を停止する
　　ボタンの表示を「自動」に変更
というふうにプログラムが動くようにしたい。
ボタンをクリックしたときのイベントであるので、btnAuto をダブルクリックして生成される関数 btnAuto_Click に次のプログラムを入力する。



```
private void btnAuto_Click(object sender, EventArgs e)
    {
        if (btnAuto.Text == "自動")
        {
            btnAuto.Text = "停止";
            timer1.Interval = 10;     // 単位は [ms]
            timer1.Start();
        }
        else
        {
            btnAuto.Text = "自動";
            timer1.Stop();
        }
    }
```

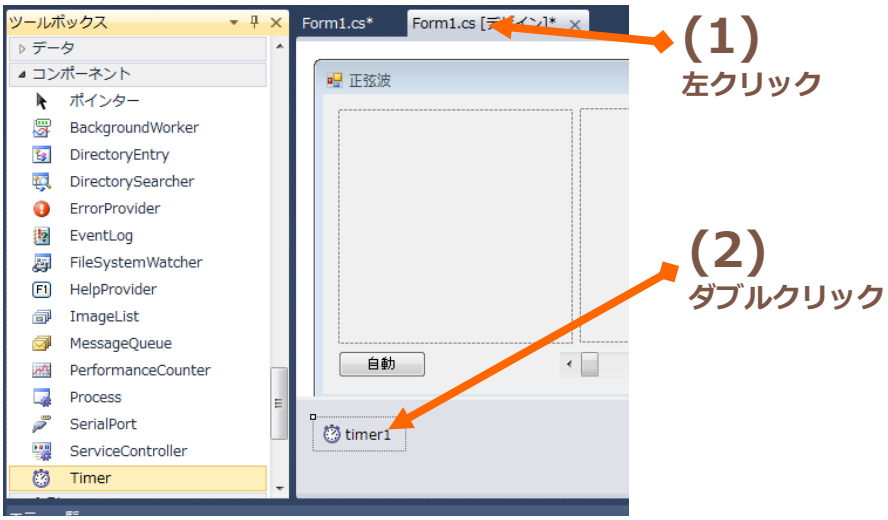
この部分を入力

10 タイマーが動作したときの処理

タイマーが動作したとき、青色の丸が表示される位相 Theta を増やし、再描画すれば青丸が移動したアニメーションが実現できる。

また、スクロールバーも同じように移動するようにする。 スクロールバーの移動はプロパティ Value に値を代入すれば行われる。しかし、最大値 Maximum を超えるとエラーになるので、最大値 Maximum を超えると 最初値 Minimum になるように処理する。

(1) Form1.cs [デザイン] をクリックして、コード入力のタグからフォームのデザインするタグへ戻る
(2) timer1 をダブルクリック



自動的に生成された timer1_Tick 関数に次の部分を入力する。

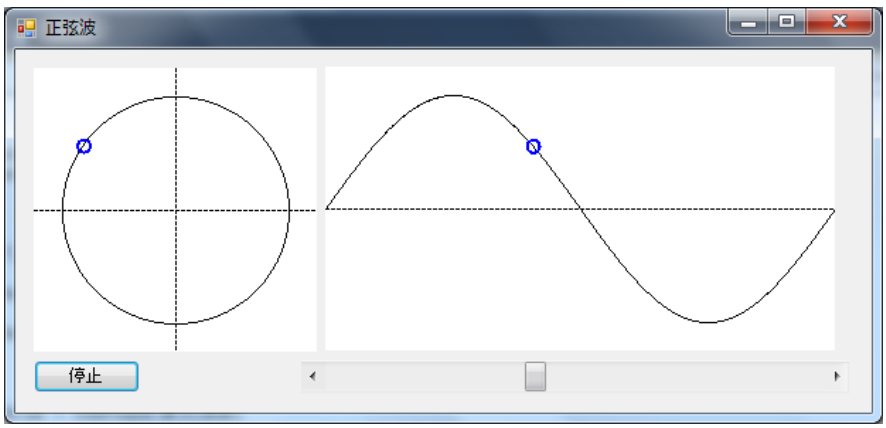
```
private void timer1_Tick(object sender, EventArgs e)
    {
        if (hsbPhase.Value < hsbPhase.Maximum)
        {
            hsbPhase.Value += 1;
        }
        else
        {
            hsbPhase.Value = hsbPhase.Minimum;
        }
        Theta = hsbPhase.Value;
        Refresh();

    }
```

この部分を入力

11 完成

これで今回のプログラムは完成
以下の動作が行われるか確認

- ■プログラムを実行すると、円と正弦波が表示される
  （青丸 は位相 0 の場所に表示される）
- ■水平スクロールバーを移動させると、それに合わせて円と正弦波上の青丸が移動する
- ■「自動」ボタンをクリックすると青丸が移動を開始する
- ■位相が最大値になったら、最小値になって移動を続ける
  （正弦波の青丸が右端までいったら左端から出てくる）
- ■ 「停止」ボタンをクリックすると、青丸の移動が停止する。



１２ まとめ

- フォームの再描画イベントを使う
  - ウィンドウが再表示されたときに発生するイベント「再描画」
  - 強制的に再描画イベントを起こさせるには Refresh();
  - タイマーと組み合わせるとアニメーションが簡単に作ることができる
- 水平スクロールバー
  - マウスによる操作で値を簡単に変更できるコントロール
  - 変更したときにイベントが生じる
  - プロパティ Value を変更すると、スクロールバーも移動する
  - 垂直スクロールバー VScrollBar というものもある
- タイマー
  - 一定の時間間隔でなにかをやりたいとき（アニメーションなど）に使う。
  - フォーム上には表示されないコントロール
  - 時間間隔は プロパティ Interval で決まる。単位は ms = ミリ秒
  - メソッド Start() で利用開始、Stop() で利用停止
- 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）
  - HScrollBar コントロール（数値をマウスでコントロール、水平方向）**NEW!**
  - Timer コントロール（一定の時間間隔でイベントを発生）**NEW!**

１３ 追加課題

1 正弦波のほうに目盛として、30度ごとに縦の点線を描く
また、円のほうも 30度づつの変化がわかるように点線で円を区切る半径を描く

2 位相が 120 度、240度 ずれている正弦波を 2 本追加し、それぞれ、円上と正弦波上に位相の変更と連動する赤丸と緑丸を追加する。

3 振幅（円の半径）を変更するスクロールバーを追加し、振幅（円の半径）を変更可能にする