Visual Basic 応用講座

CONTENTS

オブジェクト指向

オブジェクト指向とは、機能ごとにグループ分けしたプログラムのコードを独立させて、使い回しできるようにするという考え方です。

複数のデータ（変数）とそのデータに対する処理命令群を、オブジェクトというひとつのかたまりとし、それらを組み合わせてプログラムを構築していきます。

プログラマは必要に応じて、使いたいオブジェクトを選び、オブジェクトが持っているデータや処理命令を利用していきます。そうすることによって、1からコードを書くという手間を省くことができるのです。

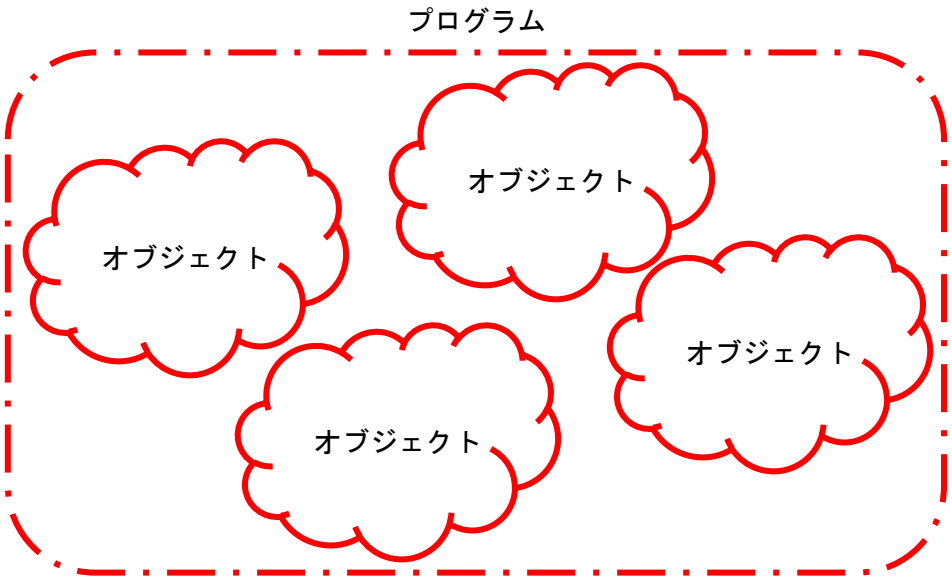
既に存在するオブジェクトについては、利用においてその内部構造や動作原理の詳細を知る必要はなく、外部から呼び出しを行えば機能するため、特に大規模なソフトウェア開発において有効な考え方であるとされています。

たとえばゲーム機はコントローラを動かすと画面の中で反応があります。

内部でどんな処理をしているのか知らなくても、操作をすることができるということです。

オブジェクトも同様で、内部で行われている動作の定義を知らなくても、使い方を知っていればプログラムは動きます。

つまり、「用意されているものを使うことで、楽にプログラムを組むことができる」これがオブジェクト指向の利点です。



クラスとオブジェクト

あらかじめ用意されている大量のオブジェクトを、プログラムの起動時に全てメモリ上に置いておくのは非効率です。数が膨大ですし、その機能のすべてをいつも使うわけではありません。

また、あるオブジェクトの大部分が同じでちょっとした値の違いや一部分のみ動作の違う処理を実現するために、別のオブジェクトを１から新しく作るのも大変な労力である上、きりがありません。

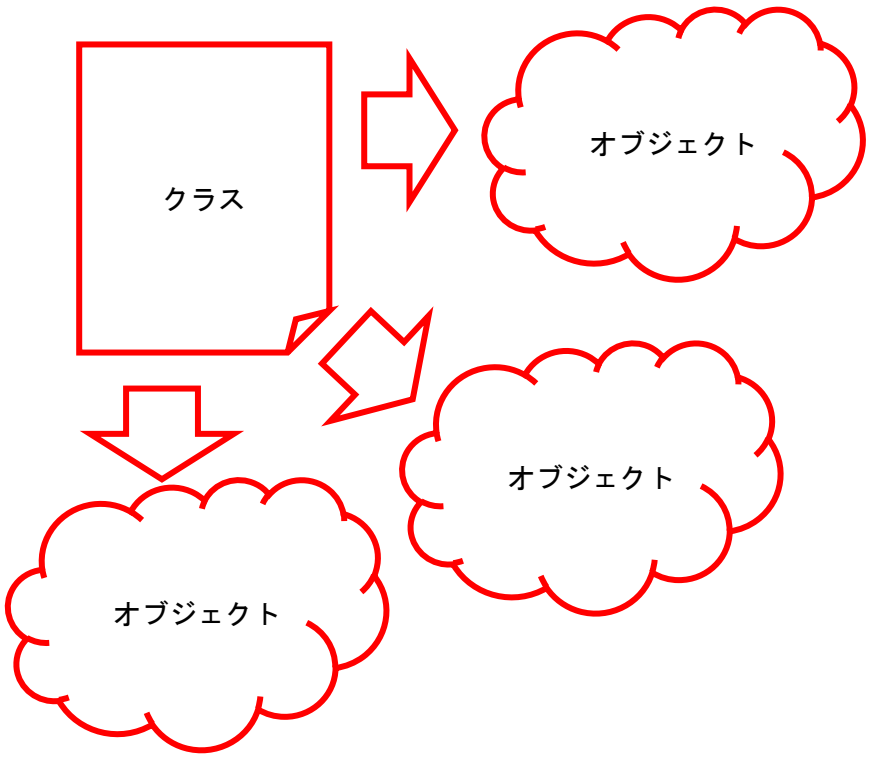
そこで、オブジェクトをメモリに直接置くのではなく、必要になったときにメモリに置けるようなシステムが必要です。さらに処理を付け加えたり、改良したりといった作業が簡単に行える汎用性も必要です。

先に学んだ基本データ型も、いきなり変数がメモリ上に存在するのではなく、まずは型が存在して、変数を新しく使う都度、その型で宣言をしました。そうすることで、同じ型の変数をいくつもひとつのプログラムで利用することができました。

そして、その都度値を設定することができました。

オブジェクトにも、同じことが言えます。

つまり、最初にオブジェクトの型を用意しておき、必要になったらその型でオブジェクトを作成するのです。そのためのオブジェクトの型を『クラス』といいます。また、オブジェクトのことを「インスタンス」とも言い、オブジェクトを生成することを「インスタンス化」と言います。



クラスの構成

クラスは基本データ型と違いさまざまな機能をもっています。基本データ型はデータを一つしか扱えませんが、クラスは複数のデータをまとめて扱い、さらに処理命令を実行することができます。まずはクラスの構成要素を確認しましょう。

フィールド

クラス内で宣言された変数のことを『フィールド』といいます。クラス内で扱う値を格納するための変数を定義します。ひとつのクラスに複数のフィールドを定義できます。

メソッド

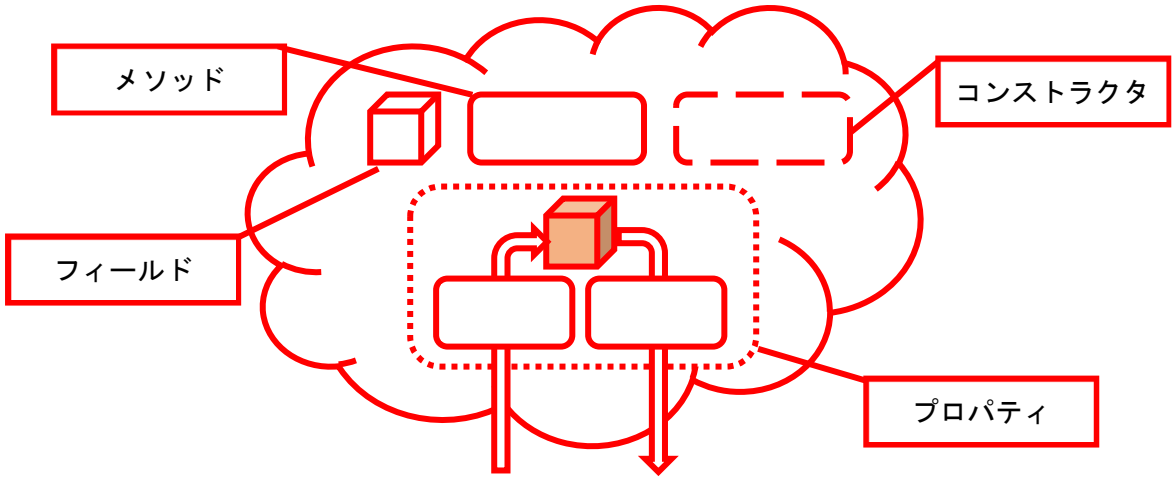
クラス内で定義された処理命令のことを『メソッド』といいます。ひとつのクラスに複数のメソッドを定義できます。

コンストラクタ

コンストラクタとはクラスからオブジェクトを生成する際に、自動的に必ず実行される処理です。オブジェクト内の初期状態を決める処理を記述します。主に、フィールドの初期値の代入処理を記述します。

プロパティ

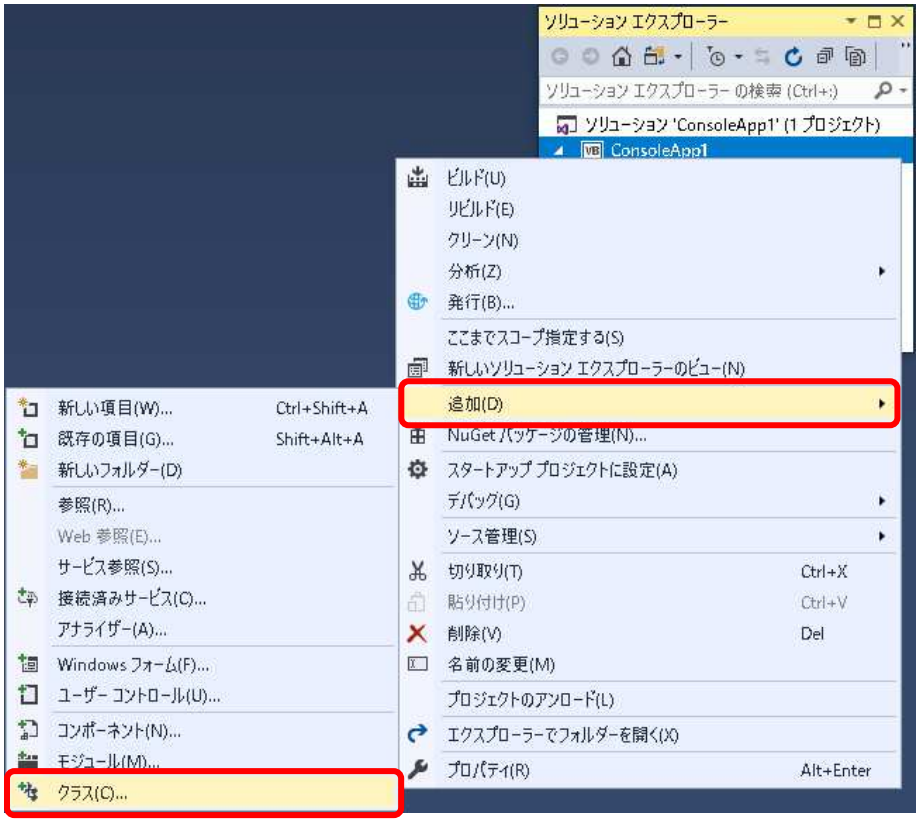
フィールドと同様に、クラス内で扱う値を格納するものですが、他のクラスからは直接値を読み込んだり書き込んだりさせず、専用のプロシージャを介してデータのやり取りを行うものです。そのプロシージャのことを「アクセサ」と言います。詳細は後ほど取り上げます。



では、さっそくクラスを利用したプログラムを確認して行きましょう。

クラス（ファイル）の作成方法

ソリューションエクスプローラーのプロジェクトを右クリックし、「追加」から「クラス」を選択します。



「クラス」を選択します。名前は任意です。「追加」をクリックします。

実装 Sample01

【Student.vb】

```
1.  Public Class Student
2.
3.      Dim name As String
4.      Dim score As Integer
5.
6.      Sub SetData(tmpName As String, tmpScore As Integer)
7.
8.          name = tmpName
9.          score = tmpScore
10.
11.     End Sub
12.
13.     Function Disp() As String
14.
15.         Return "名前：" & name & "  点数：" & score
16.
17.     End Function
18.
19. End Class
```

【Module1.vb】

```
1.  Module Module1
2.
3.      Sub Main()
4.
5.          Dim student1 As Student = New Student()
6.          Dim student2 As Student = New Student()
7.          Dim student3 As Student = New Student()
8.
9.          student1.SetData("鈴木一郎", 45)
10.         student2.SetData("山田花子", 90)
11.         student3.SetData("田中次郎", 70)
12.
13.         Console.WriteLine( student1.Disp() )
14.         Console.WriteLine( student2.Disp() )
15.         Console.WriteLine( student3.Disp() )
16.
17.     End Sub
18.
19. End Module
```

【実行結果】

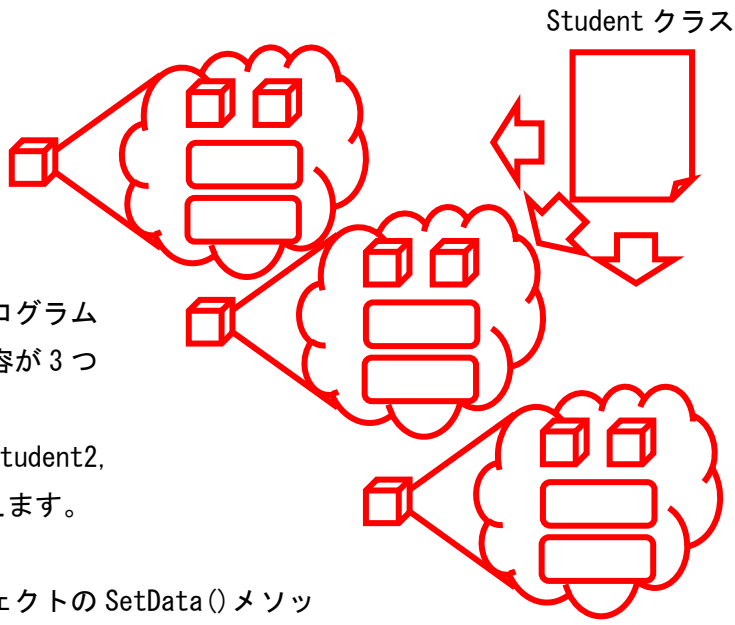
```
名前：鈴木一郎　点数：45
名前：山田花子　点数：90
名前：田中次郎　点数：70
```

解説

上記の記述で Student クラスのオブジェクトを三つ生成しています。それぞれ、student1, student2,student3 という名前の変数で扱おうとしています。

```
5.          Dim student1 As Student = New Student()
6.          Dim student2 As Student = New Student()
7.          Dim student3 As Student = New Student()
```

この状態を図で表すと、右のようになります。



クラスの内容はオブジェクトになって初めて利用可能な状態になります。オブジェクトを 3 つ生成しているので、実行中のプログラム上には、クラス内に定義された内容が 3 つずつ存在します。

それぞれの区別は、student1, student2, student3 という変数によって行えます。

下記の部分はそれぞれのオブジェクトの SetData()メソッドおよび Disp()メソッドを呼び出しています。

```
9.          student1.SetData("鈴木一郎", 45)
10.         student2.SetData("山田花子", 90)
11.         student3.SetData("田中次郎", 70)
12.
13.         Console.WriteLine( student1.Disp() )
14.         Console.WriteLine( student2.Disp() )
15.         Console.WriteLine( student3.Disp() )
```

SetData()メソッドは Sub プロシージャですが、Disp()メソッドは Function プロシージャです。戻り値は String として定義されているので、それが Console.Write()メソッドによって出力されます。

今までとは違い、メソッド名だけでメソッドを呼び出せません。必ず、どのオブジェクトのメソッドであるかを記述します。それは、メソッド名だけでは、どのオブジェクトのメソッドであるか判断できないからです。

値型と参照型

オブジェクトを扱う変数は（オブジェクト）参照変数と言い、変数の中に値が直接入っているわけではありません。オブジェクトにたどり着くための、そのオブジェクトの場所の情報が入っています。Integer など、数値などのデータを直接扱うものは値型と言うのに対して、オブジェクトを扱うものは参照型と言います。

主な型の分類です。

値型	参照型
Byte , Short , Integer , Long , Single , Double , Boolean , Date	String

補足　Integer は値型か参照型か

Integer は数値を直接変数に入れて扱う、値型であるという説明がありましたが、Integer.Parse()メソッドという記述がありました。これは Integer の中の Parse メソッドを呼び出しています。メソッドを持っているということは、Integer はクラスであり、それを扱う変数は参照型のようですが、それでは「Integer 型は値型である」と矛盾します。

これにはカラクリがあります。Integer は値型であり、メソッドを持っているのは Int32 というクラスです。そして、Integer.Parse()と記述した場合、自動的に Int32.parse()と認識されるようになっています。

練習問題 Practice01

四角形の幅（横）と高さ（縦）を管理し、面積を算出できる Rectangle クラスを作成してください。

クラス名	Rectangle

フィールド

変数名	型	内容
width	Integer 型	横幅の情報を管理するための変数
height	Integer 型	高さの情報を管理するための変数

メソッド

プロシージャ	Sub		
メソッド名	SetWidth	戻り値の型	（なし）
引数	Integer 型		
引数で指定された値をフィールド変数 width へ代入する			

プロシージャ	Sub		
メソッド名	SetHeight	戻り値の型	（なし）
引数	Integer 型		
引数で指定された値をフィールド変数 height へ代入する			

プロシージャ	Function		
メソッド名	GetArea	戻り値の型	Integer 型
引数	（なし）		
フィールド変数 width および height をもとに面積を求めて返す			

また、そのクラスの動作を確認するための main()メソッドを持つモジュールを作成して実行してください。

【Module1.vb】

```
Module Module1

    Sub Main()

        Rectangle クラスのオブジェクトを生成する

        Console.Write("四角形の横幅を入力してください。")

        入力された数字を横幅に設定する

        Console.Write("四角形の高さを入力してください。")

        入力された数字を高さに設定する

        面積を算出して表示させる

    End Sub

End Module
```

【実行結果例】

```
四角形の横幅を入力してください。8
四角形の高さを入力してください。5
四角形の面積は 40
```


練習問題 解答例 Practice01

【Module1.vb】

```
Module Module1

    Sub Main()

        'Rectangle クラスのオブジェクトを生成する
        Dim rect As Rectangle = New Rectangle()

        Console.Write("四角形の横幅を入力してください。")

        '入力された数字を横幅に設定する
        rect.SetWidth( Integer.Parse(Console.ReadLine()) )

        Console.Write("四角形の高さを入力してください。")

        '入力された数字を高さに設定する
        rect.SetHeight( Integer.Parse(Console.ReadLine()) )

        '面積を算出し、表示させる
        Console.WriteLine("四角形の面積は" & rect.GetArea())

    End Sub

End Module
```

【Rectangle.vb】

```
Public Class Rectangle

    Dim width As Integer
    Dim height As Integer

    Sub SetWidth(tmpWidth As Integer)

        width = tmpWidth

    End Sub

    Sub SetHeight(tmpHeight As Integer)

        height = tmpHeight

    End Sub

    Function GetArea() As Integer

        Return width * height

    End Function

End Class
```

コンストラクタ

コンストラクタとはクラスからオブジェクトを生成する際に、自動的に必ず実行される処理です。オブジェクト内の初期状態を決める処理を記述します。主に、フィールドの初期値の代入処理を記述します。

オブジェクト指向では、オブジェクトは生成された段階ですでに使える状態であることが望ましいです。値を設定するだけではありません。

例えば、ファイルを管理するクラスであれば、ファイルを開くところまでを行うことで、すぐにファイルの中身が扱えるようにしたり、データベースを管理するクラスであれば、データベースへ接続するところまで行うことで、データの中身をすぐに扱えるようにしたり、そのオブジェクトが最低限使える状態にする処理を記述する場所がコンストラクタです。

また、何度でも呼び出せるメソッドとは異なり、コンストラクタはオブジェクトが生成されるタイミングで一度だけ呼び出され、そのオブジェクトでは、その後二度と呼び出せません。

別のオブジェクトを生成する場合には、そのオブジェクトに対するコンストラクタが実行されます。

書式

コンストラクタの記述はメソッドの記述と全体的に似ていますが、名前は「New」で固定です。任意の名前は付けられません。

また、コンストラクタには戻り値を設定できませんので、Function プロシージャではなく、Sub プロシージャである必要があります。

```
Sub New（引数宣言）
    コンストラクタで行う処理
    コンストラクタで行う処理
    ・・・
End Sub
```

コンストラクタを呼び出し、オブジェクトを生成するときの書式は下記のとおりです。

```
    Dim 変数名 As クラス名 = New クラス名（引数）
```

引数の定義は任意ですが、メソッドの引数と同様に、呼び出し時には一致させる必要があります。

実装 Sample02

【Student.vb】

```
1.  Public Class Student
2.
3.      Dim name As String
4.      Dim score As Integer
5.
6.      Sub SetData(tmpName As String, tmpScore As Integer)
7.
8.          name = tmpName
9.          score = tmpScore
10.
11.     End Sub
12.
13.     Function Disp() As String
14.
15.         Return "名前：" & name & "  点数：" & score
16.
17.     End Function
18.
19.     Sub New(tmpName As String, tmpScore As Integer)
20.
21.         name = tmpName
22.         score = tmpScore
23.
24.     End Sub
25.
26. End Class
```

【Module1.vb】

```
1.  Module Module1
2.
3.      Sub Main()
4.
5.          Dim student As Student = New Student("鈴木一郎", 45)
6.
7.          Console.WriteLine( student.Disp() )
8.
9.          student.SetData("山田花子", 90)
10.
11.         Console.WriteLine( student.Disp() )
12.
13.     End Sub
14.
15. End Module
```

【実行結果】

```
名前：鈴木一郎　点数：45
名前：山田花子　点数：90
```

解説

まずはコンストラクタの部分です。

```
19.         Sub New(tmpName As String, tmpScore As Integer)
20.
21.             name = tmpName
22.             score = tmpScore
23.
24.         End Sub
```

今回のコンストラクタでは 2 つの引数を受け取り、オブジェクトの情報としてすぐに使えるようにフィールドにセットしています。

```
5.              Dim student As Student = New Student("鈴木一郎", 45)
6.
7.              Console.WriteLine(student.Disp())
8.
9.              student.SetData("山田花子", 90)
10.
11.             Console.WriteLine(student.Disp())
```

フィールドに値が設定されているので、すぐに Disp()メソッドを利用できます。そして、オブジェクト内の SetData()メソッドを利用してフィールドの更新が可能です。

このように、メソッドは何度でも呼び出すことができますが、コンストラクタは 1 つのオブジェクトに対して一度だけしか呼び出せません。

練習問題 Practice02

以前の練習問題で作成した四角形（Rectangle）クラスにコンストラクタを追加します。コンストラクタの詳細は次のとおりです。

コンストラクタ	
プロシージャ	Sub
引数	Integer 型 , Integer 型
第一引数：横幅、第二引数：高さ 上記情報を受け取り、自身のオブジェクトの各フィールドへ格納する	

また、そのクラスの動作を確認するための main()メソッドを持つモジュールは次のとおりです。

```
Module Module1

    Sub Main()

        必要であれば必要な分だけ変数宣言を記述する

        Console.Write("四角形の横幅を入力してください。")

        入力された数字を横幅として取得する

        Console.Write("四角形の高さを入力してください。")

        入力された数字を高さとして取得する

        Rectangle クラスのコンストラクタを利用してオブジェクトを生成する

        面積を算出して表示させる

    End Sub

End Module
```

【実行結果例】

```
四角形の横幅を入力してください。8
四角形の高さを入力してください。5
四角形の面積は 40
```

練習問題 解答例 Practice02

【Module1.vb】

```
Module Module1

    Sub Main()

        Dim w As Integer
        Dim h As Integer
        Dim rect As Rectangle

        Console.Write("四角形の横幅を入力してください。")

        w = Integer.Parse(Console.ReadLine())

        Console.Write("四角形の高さを入力してください。")

        h = Integer.Parse(Console.ReadLine())

        rect = New Rectangle(w, h)

        Console.WriteLine("四角形の面積は" & rect.GetArea())

    End Sub

End Module
```

【Rectangle.vb】

```
Public Class Rectangle

    Dim width As Integer
    Dim height As Integer

    Sub New(tmpWidth As Integer, tmpHeight As Integer)

        width = tmpWidth
        height = tmpHeight

    End Sub

    Sub SetWidth(tmpWidth As Integer)
        width = tmpWidth
    End Sub

    Sub SetHeight(tmpHeight As Integer)
        height = tmpHeight
    End Sub

    Function GetArea() As Integer
        Return width * height
    End Function

End Class
```

アクセス修飾子

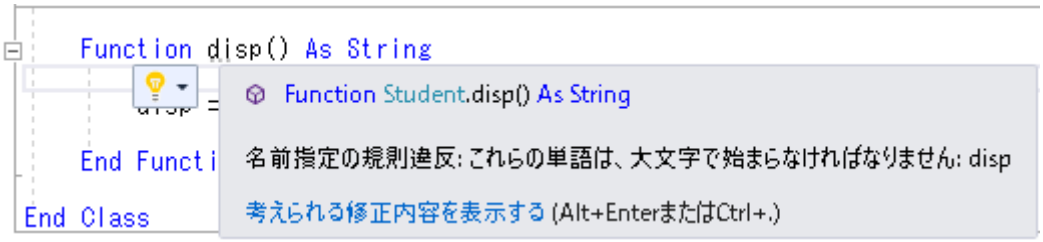
アクセス修飾子とは、クラス、フィールド、メソッドなどに付与できるキーワードで、付与されている要素の有効範囲を設定できます。下表のように 5 種類存在しますが、現段階では Public と Private の 2 種類を覚えてください。

アクセス修飾子	アクセス可能な範囲
Public	どこからでもアクセス可能
Protected	派生クラスからアクセス可能
Friend	同じプロジェクト内のクラスからアクセス可能
Protected Friend	派生クラス または 同じプロジェクトのクラスからアクセス可能
Private	クラス内からのみアクセス使用可能

メソッドに対してアクセス修飾子を省略した場合は Public として扱われます。

フィールドに対してアクセス修飾子を省略して Dim とした場合は Private として扱われます。

メソッドの名前の頭文字は基本的に大文字にします。小文字でもエラーにはならず、実行できますが、Visual Studio のエディタ上では大文字へ変更するように注意が促されます。



実装 Sample03

先のサンプルに、下記のようにアクセス修飾子を付与してみましょう。一部エラーになるので実行はできません。エラーになることを確認するだけです。

【Student.vb】

```
1.  Public Class Student
2.
3.      Public name As String
4.      Private score As Integer
5.
6.      Public Sub SetData(tmpName As String, tmpScore As Integer)
7.
8.          name = tmpName
9.          score = tmpScore
10.
11.     End Sub
12.
13.     Private Function Disp() As String
14.
15.         Return "名前：" & name & "  点数：" & score
16.
17.     End Function
18.
19.     Public Sub New(tmpName As String, tmpScore As Integer)
20.
21.         name = tmpName
22.         score = tmpScore
23.
24.     End Sub
25.
26. End Class
```

【Module1.vb】

```
1.  Module Module1
2.
3.      Sub Main()
4.
5.          Dim student As Student = New Student("鈴木一郎", 45)
6.
7.          student.name = "山田花子"
8.          student.score = 90
9.
10.         student.SetData("田中次郎", 70)
11.         student.Disp()
12.
13.     End Sub
14.
15. End Module
```

解説

```
7.              student.name = "山田花子"
8.              student.score = 90
9.
10.             student.SetData("田中次郎", 70)
11.             student.Disp()
```

Module1.vb の 8 行目がエラーになりました。フィールド変数 score が Private で宣言されているので、この変数に対して、ほかのクラスからのアクセスができなくなっていることが確認できます。また、11 行目も同様に、Disp()メソッドも Private で宣言されているので、アクセスできなくなっています。

確認が終了したら、Private になっている部分を Public に変更しておいてください。

【Student.vb】

```
1.  Public Class Student
2.
3.      Public name As String
4.      Public score As Integer
5.
6.      Public Sub SetData(tmpName As String, tmpScore As Integer)
7.
8.          name = tmpName
9.          score = tmpScore
10.
11.     End Sub
12.
13.     Public Function Disp() As String
14.
15.         Return "名前：" & name & "  点数：" & score
16.
17.     End Function
18.
19.     Public Sub New(tmpName As String, tmpScore As Integer)
20.
21.         name = tmpName
22.         score = tmpScore
23.
24.     End Sub
25.
26. End Class
```


プロパティ

フィールドと同様に、クラス内で扱う値を格納するものですが、他のクラスからは直接値を読み込んだり書き込んだりさせず、専用のプロシージャを介してデータのやり取りを行うものです。

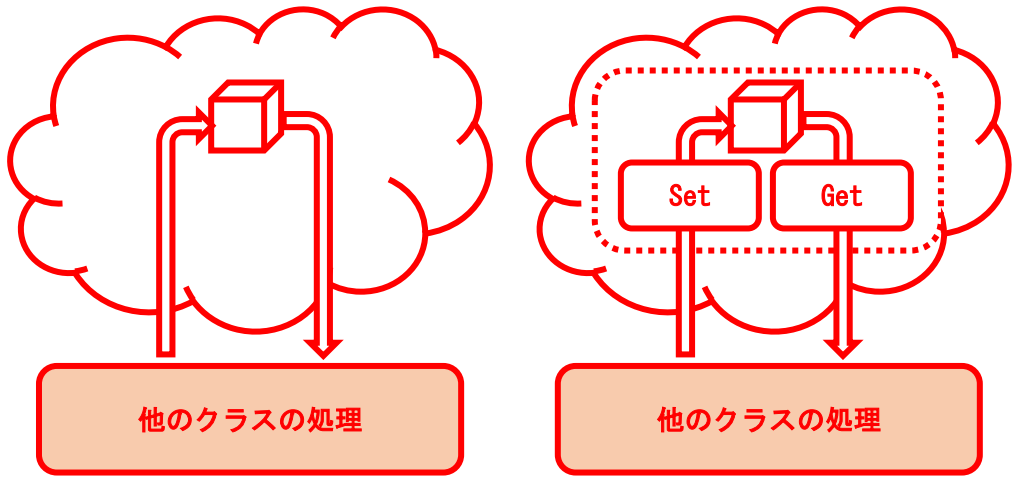
フィールドはアクセス修飾子を Public に設定することで、ほかのクラスから直接値を扱うことが可能でしたが、プロパティはアクセス修飾子を Private に設定し、ほかのクラスから直接値を扱えなくします。直接扱えなくする理由は、データの保護のためです。

Public で宣言されたフィールドは直接値が扱えて便利ですが、不正な入力も許してしまいます。そこで、専用プロシージャを介してデータのやり取りを行わせます。そのプロシージャのことを「アクセサ」と言います。

アクセサを用意することで、データのチェックなどが可能になり、不正な扱いを防止できます。

アクセサはデータを格納するための Set と、データを取得するための Get があります。Set は Set（プロパティ）プロシージャ、Set アクセサ、Setter（セッター）とも呼ばれ、Get は Get（プロパティ）プロシージャ、Get アクセサ、Getter（ゲッター）とも呼ばれます。

このように、内部のデータを外部から隠ぺいすることをカプセル化と言います。



書式

ある変数に対するプロパティプロシージャの定義方法は下記のとおりです。

```
Private 変数名 As データ型

Public Property プロパティ名 As データ型
    Get
        Return 変数名
    End Get
    Set(value As データ型)
        変数名 = value
    End Set
End Property
```

プロパティ名の頭文字は大文字にします。

呼び出す側の記述の例は次のとおりです。

```
オブジェクト参照変数．プロパティ名 = データ
変数 = オブジェクト参照変数．プロパティ名
```

Private で宣言されているフィールドですが、プロパティを利用すると、あたかもフィールド変数を直接扱っているかのように記述できます。実際はデータを設定する場合は Set アクセサが、データを参照する場合は Get アクセサが呼ばれています。

実装 Sample04

フィールド変数 score をプロパティに変更します。フィールド変数 name はまだ変更しません。

【Student.vb】

```
1.  Public Class Student
2.
3.      Public name As String
4.      Private _score As Integer
5.
6.      Public Property Score As Integer
7.          Get
8.              Return _score
9.          End Get
10.         Set(value As Integer)
11.             _score = value
12.         End Set
13.     End Property
14.
15.     Public Sub SetData(tmpName As String, tmpScore As Integer)
16.
17.         name = tmpName
18.         Score = tmpScore
19.
20.     End Sub
21.
22.     Public Function Disp() As String
23.
24.         Return "名前：" & name & "  点数：" & Score
25.
26.     End Function
27.
28.     Public Sub New(tmpName As String, tmpScore As Integer)
29.
30.         name = tmpName
31.         Score = tmpScore
32.
33.     End Sub
34. End Class
```

【Module1.vb】

```
1.  Module Module1
2.
3.      Sub Main()
4.
5.          Dim student As Student = New Student("鈴木一郎", 45)
6.          Dim num As Integer
7.
8.          student.Score = 90
9.          num = student.Score
10.
11.         Console.WriteLine( student.Disp() )
12.         Console.WriteLine( num )
13.
14.     End Sub
15.
16. End Module
```

このように、プロパティの記述は決まった形式の部分が多く、パターン化されています。フィールドが複数存在するとき、すべてのフィールドでこの記述をするのは大変です。そこで、簡単にコードを生成する方法を紹介しますので、フィールド変数 name に対してプロパティを設定してみましょう。

フィールドの変数名のところにカーソルを置くと行番号の隣にアイコンが表示され、クリックするとメニューが表示されます。

2 種類の「フィールドのカプセル化」の方法があります。マウスを当てると変更前後のソースコードが右側に表示されるので変化が分かりやすいです。

補足

2 種類の「フィールドのカプセル化」の違いは、フィールド変数を扱っているプロシージャ内の変数名も同時に変更するか否かです。今回はどちらを使っても最終的には同じ結果になります。

メニューの上側を選択した場合、メソッド内の変数名の頭文字が大文字に変換され、Name に変わります。

```
    Private _name As String

    Public Property Name As String
        Get
            Return _name
        End Get
        Set(value As String)
            _name = value
        End Set
    End Property

    Public Sub SetData(tmpName As String, tmpScore As Integer)

        Name = tmpName
        Score = tmpScore

    End Sub
```

メニューの下側を選択した場合、メソッド内の変数名は name（頭文字が小文字）のままですが、プロパティの自動挿入でフィールド名が name から_name に変わり、Name プロパティが生成され、頭文字が小文字の name の名前を持つものが存在しなくなります。Visual Studio は大文字と小文字の差しかない name を Name のタイプミスと判断し、name のフィールド名を保持させても、Name と自動変換されます。

```
    Private _name As String

    Public Property Name As String
        Get
            Return _name
        End Get
        Set(value As String)
            _name = value
        End Set
    End Property

    Public Sub SetData(tmpName As String, tmpScore As Integer)

        name = tmpName
        Score = tmpScore

    End Sub
```

後に自動変換される

解説

下記は Student クラスのフィールド変数 score をプロパティにした時のソースコードです。プロパティ名は頭文字を大文字にします。単純にそうすると、プロパティ名とフィールド名が大文字と小文字の違いしかなくなってしまい、エラーになります。それを避けるために、フィールド名の先頭に _ アンダースコアを追加します。

命名の規則を守っていれば別の名前も付けられますが、フィールド変数 name のプロパティの自動生成でも確認できるとおり、この書き方が一般的な名前の付け方です。

【Student.vb】

```
4.          Private _score As Integer
5.
6.          Public Property Score As Integer
7.              Get
8.                  Return _score
9.              End Get
10.             Set(value As Integer)
11.                 _score = value
12.             End Set
13.         End Property
```

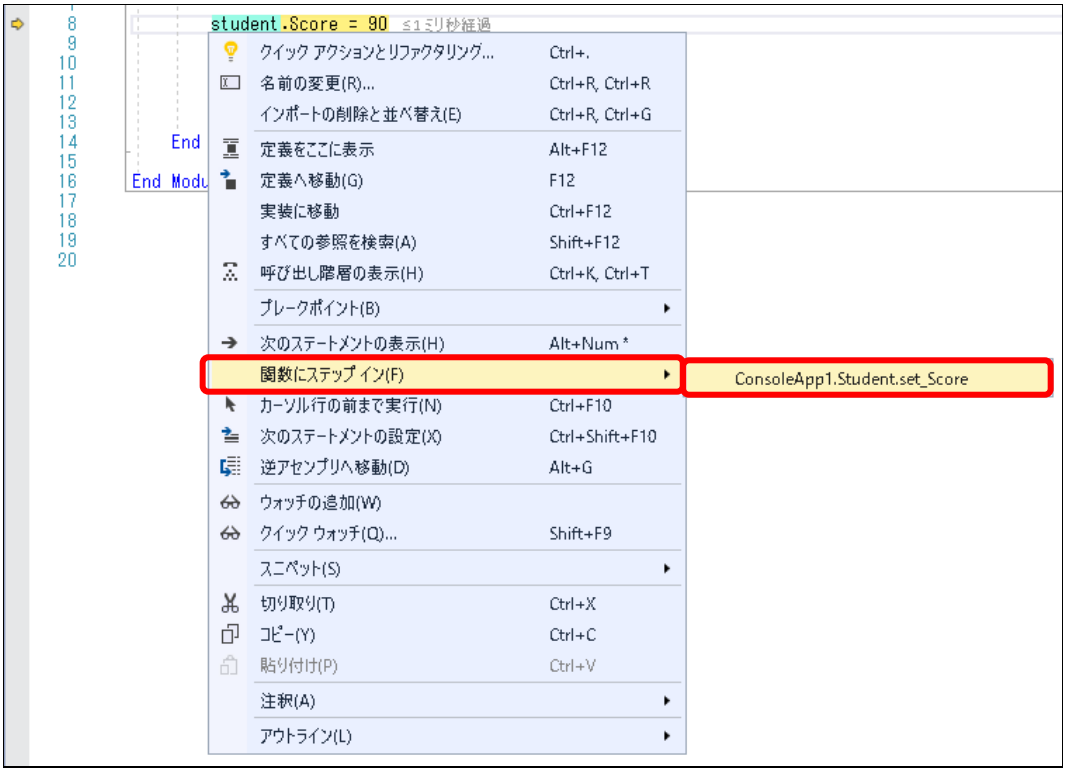
下記はプロパティ Score に対してデータを設定および取得をしている部分です。この部分だけ見るとフィールド変数を直接扱っているように見えますが、実際は Get アクセサおよび Set アクセサを介してデータがやり取りされています。

【Module1.vb】

```
8.              student.Score = 90
9.              num = student.Score
```

8 行目および 9 行目でステップイン実行すると、Get アクセサおよび Set アクセサの記述が実行されていることが確認できます。

ステップイン実行時に、単純にステップインできない旨のメッセージが表示されますが、該当行を右クリックし、「関数にステップイン」を選択すると Get アクセサおよび Set アクセサの中へステップインできます。



練習問題 Practice03

Student クラスのプロパティに対して、次のような設定をしましょう。

また、実行確認のために Main()メソッドを編集し、名前と点数の入力が可能なようにしましょう。

プロパティ	条件
Name	1 文字以上、10 文字以下であること 条件を満たさない入力が行われたら不正であることを通知し、内容は変更しない
Score	0 以上、100 以下であること 条件を満たさない入力が行われたら不正であることを通知し、内容は変更しない

【実行結果例 1】

```
初期値      名前：鈴木一郎　点数：45

名前を入力してください。山田花子
点数を入力してください。90

変更後      名前：山田花子　点数：90
```

【実行結果例 2】

```
初期値      名前：鈴木一郎　点数：45

名前を入力してください。山田花子山田花子山田花子
名前の入力が不正です
点数を入力してください。250
点数の入力が不正です

変更後      名前：鈴木一郎　点数：45
```

文字数

文字列の長さ（文字数）を得るには、String クラスの Length プロパティを使用します。

String もクラスなので、さまざまなメソッドやプロパティを持っています。その中でも、Length プロパティは文字数を管理しているプロパティです。ただし、このプロパティは読み取り専用で、変更はできません。

```
Dim str As String = "ABC"
Dim len As Integer = str.Length
Console.Write( len )                    '3と表示される
```

練習問題 解答例 **Practice03**

【Module1.vb】

```
Module Module1

    Sub Main()

        Dim student As Student = New Student("鈴木一郎", 45)

        Console.WriteLine("初期値    " & student.Disp())
        Console.WriteLine()

        Console.Write("名前を入力してください。")
        student.Name = Console.ReadLine()

        Console.Write("点数を入力してください。")
        student.Score = Integer.Parse(Console.ReadLine())

        Console.WriteLine()
        Console.WriteLine("変更後    " & student.Disp())

    End Sub

End Module
```

【Student.vb】

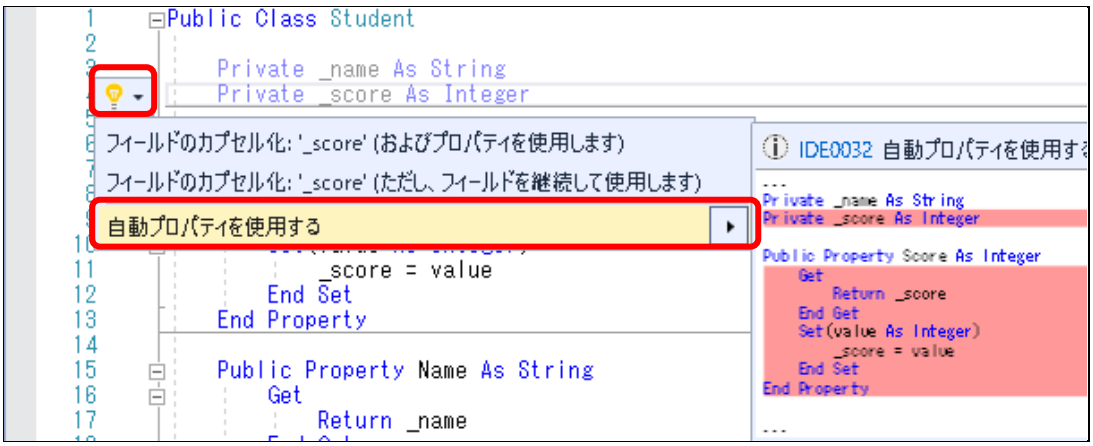
```
Public Class Student

    Private _name As String
    Private _score As Integer

    Public Property Score As Integer
        Get
            Return _score
        End Get
        Set(value As Integer)
            If value >= 0 And value <= 100 Then
                _score = value
            Else
                Console.WriteLine("点数の入力が不正です")
            End If
        End Set
    End Property

    Public Property Name As String
        Get
            Return _name
        End Get
        Set(value As String)
            If value.Length >= 1 And value.Length <= 10 Then
                _name = value
            Else
                Console.WriteLine("名前の入力が不正です")
            End If
        End Set
    End Property

    Public Sub SetData(tmpName As String, tmpScore As Integer)

        Name = tmpName
        Score = tmpScore

    End Sub

    Public Function Disp() As String

        Return "名前：" & Name & "  点数：" & Score

    End Function

    Public Sub New(tmpName As String, tmpScore As Integer)

        Name = tmpName
        Score = tmpScore

    End Sub

End Class
```


プロパティ表記の簡略化（自動プロパティ）

先ほどまでは下記のように、フィールド変数の宣言とプロパティ(Get/Set)の宣言がありました。

```
4.          Private _score As Integer
5.
6.          Public Property Score As Integer
7.              Get
8.                  Return _score
9.              End Get
10.             Set(value As Integer)
11.                 _score = value
12.             End Set
13.         End Property
```

プロパティ設定済みのフィールド変数に対して、再びカーソルを合わせると、先ほどとは異なる電球のアイコンが表示されます。「自動プロパティを使用する」を実行すると、プロパティの記述が簡略化できます。



実行すると、上記の 9 行分が下記の 1 行だけになります。

```
        Public Property Score As Integer
```

Get アクセサおよび Set アクセサ内で条件判定などの処理を行わず、単純に値を出し入れするだけであれば、この 1 行の記述でもかまいません。(最初からこの一行を記述しても問題ありません。)

ただし、値のチェック処理などが記述されている場合は簡略化できず、「自動プロパティを使用する」の選択肢は表示されません。

前述のように、容易に宣言できるからといって、すべてのフィールド変数をプロパティにすべきではありません。そもそもカプセル化の発想として、値を直接扱わせず、アクセスに制限をかけたり値のチェックができたりすることで、データの安全性を高めるためのものです。それが、あたかも Public 変数を扱うかのように記述できてしまうプロパティは注意して利用する必要があります。

練習問題 Practice04

以前の練習問題で作成した四角形（Rectangle）クラスを大幅に変更します。下記の内容を持つクラスとなるように作成してください。記述がないメソッドに関しては削除してください。

クラス名	Rectangle	アクセス修飾子	Public

フィールド

プロパティ名	型	内容
Width	Integer 型	横幅の情報を管理するためのプロパティ
Height	Integer 型	高さの情報を管理するためのプロパティ

コンストラクタ

プロシージャ	Sub	アクセス修飾子	Public
引数	Integer 型 , Integer 型		
第一引数：横幅、第二引数：高さ 上記情報を受け取り、自身のオブジェクトの各フィールドへ格納する			

メソッド

プロシージャ	Function	アクセス修飾子	Public
メソッド名	GetArea	戻り値の型	Integer 型
引数	（なし）		
フィールドプロパティ Width および Height をもとに面積を求めて返す			

【Module1.vb】

```
Module Module1

    Sub Main()

        必要であれば必要な分だけ変数宣言を記述する

        Console.Write("四角形の横幅を入力してください。")

        入力された数字を横幅として取得する

        Console.Write("四角形の高さを入力してください。")

        入力された数字を高さとして取得する

        Rectangle クラスのコンストラクタを利用してオブジェクトを生成する

        面積を算出して表示させる

        Console.WriteLine()

        Console.Write("変更後の四角形の横幅を入力してください。")

        入力された数字を横幅に設定する

        Console.Write("変更後の四角形の高さを入力してください。")

        入力された数字を高さに設定する

        面積を算出して表示させる

    End Sub

End Module
```

【実行結果例】

```
四角形の横幅を入力してください。8
四角形の高さを入力してください。5
四角形の面積は 40

変更後の四角形の横幅を入力してください。7
変更後の四角形の高さを入力してください。4
変更後の四角形の面積は 28
```

練習問題 解答例 Practice04

【Module1.vb】

```
Module Module1

    Sub Main()

        Dim w As Integer
        Dim h As Integer
        Dim rect As Rectangle

        Console.Write("四角形の横幅を入力してください。")

        '入力された数字を横幅として取得する
        w = Integer.Parse(Console.ReadLine())

        Console.Write("四角形の高さを入力してください。")

        '入力された数字を高さとして取得する
        h = Integer.Parse(Console.ReadLine())

        'Rectangle クラスのオブジェクトを生成する
        rect = New Rectangle(w, h)

        '面積を算出し、表示させる
        Console.WriteLine("四角形の面積は" & rect.GetArea())

        Console.WriteLine()

        Console.Write("変更後の四角形の横幅を入力してください。")

        '入力された数字を横幅に設定する
        rect.Width = Integer.Parse(Console.ReadLine())

        Console.Write("変更後の四角形の高さを入力してください。")

        '入力された数字を高さに設定する
        rect.Height = Integer.Parse(Console.ReadLine())

        '面積を算出し、表示させる
        Console.WriteLine("変更後の四角形の面積は" & rect.GetArea())

    End Sub

End Module
```

【Rectangle.vb】

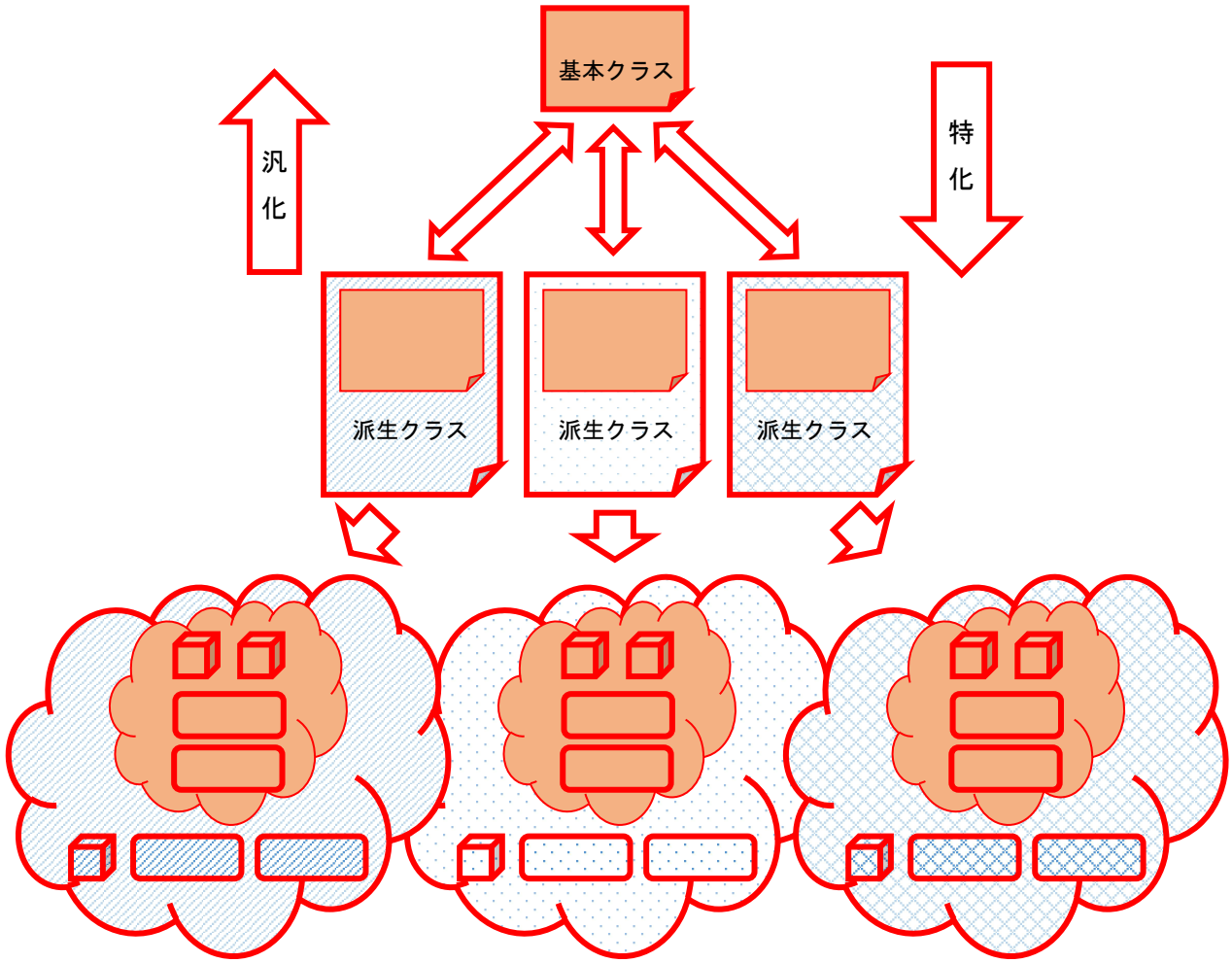
```
Public Class Rectangle

    Public Property Width As Integer
    Public Property Height As Integer

    Public Sub New(tmpWidth As Integer, tmpHeight As Integer)

        Width = tmpWidth
        Height = tmpHeight

    End Sub

    Public Function GetArea() As Integer

        Return Width * Height

    End Function

End Class
```


継承

既存クラスの機能を有効利用するために、そのクラスを引き継ぎ、機能を拡張したクラスを生成することを継承と言います。

継承元になる既存のクラスを「基本クラス／スーパークラス／親クラス」と言い、既存のクラスを継承して新たに定義したクラスを「派生クラス／サブクラス／子クラス」と言います。継承を利用することで、派生クラスは、基本クラスの内容を引き継ぐことができ、異なる部分だけをコーディングするだけでよくなります。これを差分コーディングと言います。

継承関係を構成するには、汎化と特化の 2 つのアプローチがあります。汎化とは、構造が似通った複数のクラスから共通項目を抽出して基本クラスを定義する方法です。特化とは、基本クラスをより具体化して詳細な派生クラスを定義する方法です。



書式

基本クラスは今までのクラスのように定義するだけです。派生クラスは、継承元となるクラスを指定する必要があります。

```
Class 派生クラス名
    Inherits 基本クラス名
  ・・・
End Class
```

改行が必要

この記述だけで、基本クラス内にあるフィールドやメソッドを受け継ぎ、利用可能になります。

実装 Sample05

【Module1.vb】

```
1.  Module Module1
2.
3.      Sub Main()
4.
5.          Dim car As Car = New Car()
6.
7.          car.SetData(15, 100)
8.          Console.WriteLine( car.Disp() )
9.
10.         Dim taxi As Taxi = New Taxi()
11.
12.         taxi.SetData(20, 60)
13.         taxi.Fare = 500
14.
15.         Console.WriteLine( taxi.Disp() & taxi.DispFare() )
16.
17.     End Sub
18.
19. End Module
```

【Car.vb】基本クラス

```
1.  Public Class Car
2.
3.      Public Property Fuel As Integer
4.      Public Property Speed As Integer
5.
6.      Public Sub SetData(tmpFuel As Integer, tmpSpeed As Integer)
7.
8.          Fuel = tmpFuel
9.          Speed = tmpSpeed
10.
11.     End Sub
12.
13.     Public Function Disp() As String
14.
15.         Return "  燃料：" & Fuel & "  速度：" & Speed
16.
17.     End Function
18.
19. End Class
```

【Taxi.vb】派生クラス

```
1.  Public Class Taxi
2.      Inherits Car
3.
4.      Public Property Fare As Integer
5.
6.      Public Function DispFare() As String
7.
8.          Return "  料金：" & Fare
9.
10.     End Function
11.
12. End Class
```

【実行結果】

```
燃料：15  速度：100
燃料：20  速度：60  料金：500
```

解説

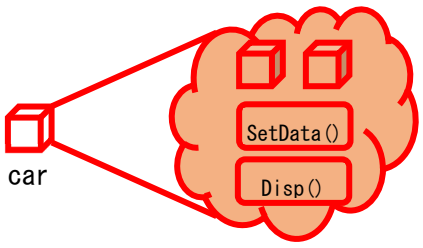
【Module1.vb】

```
5.              Dim car As Car = New Car()
6.
7.              car.SetData(15, 100)
8.              Console.WriteLine( car.Disp() )
```

上記のコードは Main()メソッド内の前半です。この部分は継承と関係なく、今まで学習してきた内容のみで記述されています。続く後半の、継承の部分との比較のために記述しています。

5 行目でオブジェクトを生成し、7 行目で SetData()メソッドを実行し、8 行目で Disp()メソッドを実行しています。

下図は 5 行目の、Car クラスをインスタンス化して変数 car で参照したときのイメージです。オブジェクトの内容は Car クラス内に定義されているものです。



次からが継承を利用した部分です。

【Module1.vb】

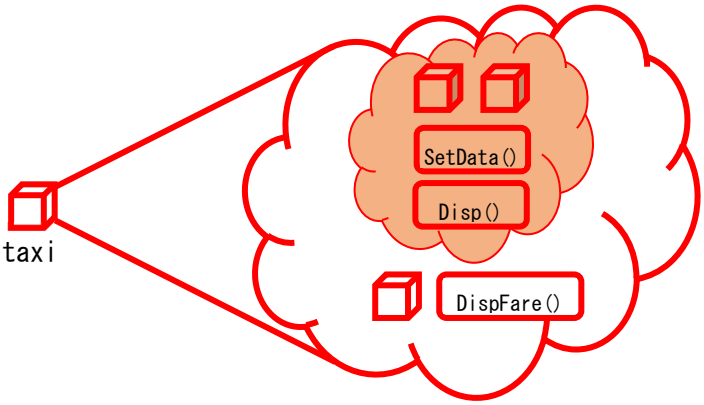
```
10.             Dim taxi As Taxi = New Taxi()
11.
12.             taxi.SetData(20, 60)
13.             taxi.Fare = 500
14.
15.             Console.WriteLine( taxi.Disp() & taxi.DispFare() )
```

10 行目で Taxi クラスのオブジェクトを生成しています。基本クラスも派生クラスも「クラス」なので、オブジェクトを生成する文法は同じです。12 行目は SetData()メソッドを呼び出していますが、Taxi クラス内に SetData()メソッドの記述はありません。しかし、Taxi クラスは下記のように Car クラスを継承しているので、Car クラス内に記述された内容を含んでいます。

【Taxi.vb】

```
1.      Public Class Taxi
2.          Inherits Car
```

右図は 10 行目の、Taxi クラスをインスタンス化し、変数 taxi で参照したときのイメージです。



Taxi クラスのオブジェクトは上図のように、基本クラスの内容を含み、Taxi クラス内に記述された内容が拡張された状態になります。

Taxi オブジェクト全体の中には SetData()メソッドも Disp()メソッドも存在しているので、どちらも呼び出すことが可能です。当然、Taxi クラスで独自に定義されている DispFare()メソッドもオブジェクト内に存在するメソッドなので、15 行目のように利用可能です。

また、フィールドおよびプロパティに関しても同様に、Car クラス内に定義されていて、Private 修飾子がついていない、アクセス可能なものであれば利用可能です。

練習問題 Practice05

GarbageTruck クラス（ゴミ収集車クラス）を作成してください。GarbageTruck クラスの詳細は次のとおりです。ただし、下記の内容は継承を考慮せず、GarbageTruck クラスで必要なすべての情報を記述しています。コーディングする際は、既存の Car クラスを利用して簡潔に表現しましょう。

クラス名	GarbageTruck	アクセス修飾子	Public

フィールド

プロパティ名	型	内容
Fuel	Integer 型	燃料の情報を管理するためのプロパティ
Speed	Integer 型	速度の情報を管理するためのプロパティ
Garbage	Double 型	ゴミの情報を管理するためのプロパティ

コンストラクタ

なし

メソッド

プロシージャ	Sub	アクセス修飾子	Public
メソッド名	SetGarbageData	戻り値の型	（なし）
引数	Integer 型 , Integer 型 , Double 型		
第一引数で受け取った値をフィールドプロパティ Fuel へ格納する。 第二引数で受け取った値をフィールドプロパティ Speed へ格納する。 第三引数で受け取った値をフィールドプロパティ Garbage へ格納する。			
プロシージャ	Function	アクセス修飾子	Public
メソッド名	DispGarbage	戻り値の型	String 型
引数	（なし）		
フィールドプロパティ Garbage をもとに表示文字列を返す。 表示例：「　ゴミ：100　」			
プロシージャ	Sub	アクセス修飾子	Public
メソッド名	CollectGarbage	戻り値の型	（なし）
引数	（なし）		
フィールドプロパティ Garbage に 0.5 加算する。			

【実行結果】

```
燃料：20　速度：60　ゴミ：10.5
```

練習問題 解答例 Practice05

【Module1.vb】

```
Module Module1

    Sub Main()

        Dim gbg As GarbageTruck = New GarbageTruck()

        gbg.SetGarbageData(20, 60, 10.5)

        Console.WriteLine(gbg.Disp() & gbg.DispGarbage())

    End Sub

End Module
```

【Car.vb】

```
Public Class Car

    Public Property Fuel As Integer
    Public Property Speed As Integer

    Public Sub SetData(tmpFuel As Integer, tmpSpeed As Integer)

        Fuel = tmpFuel
        Speed = tmpSpeed

    End Sub

    Public Function Disp() As String

        Return "  燃料：" & Fuel & "  速度：" & Speed

    End Function

End Class
```

【GarbageTruck.vb】

```
Public Class GarbageTruck
    Inherits Car

    Public Property Garbage As Double

    Public Sub SetGarbageData(tmpFuel As Integer,
                              tmpSpeed As Integer, tmpGarbage As Double)

        Fuel = tmpFuel
        Speed = tmpSpeed
        Garbage = tmpGarbage

    End Sub

    Public Function DispGarbage() As String

        Return "  ゴミ:" & Garbage

    End Function

    Public Sub CollectGarbage()

        Garbage += 0.5

    End Sub

End Class
```


オーバーライド

オーバーライドとは、基本クラスで定義しているメソッドを、派生クラス内で再定義することです。オーバーライドを利用することで、派生クラスのメソッドを優先的に使用できます。

継承することで基本クラスから引き継いだメソッドの内容が、派生クラスにふさわしくない場合、派生クラス内で再定義することで処理内容を変更できます。

先ほどのサンプルで、Car クラス（基本クラス）に Disp()メソッドがあり、Taxi クラス（派生クラス）に DispFare()メソッドがありました。Taxi クラスの情報を表示するためには、Disp()メソッドと DispFare()メソッドの両方を実行する必要があります。

【Module1.vb（一部)】

```
15.             Console.WriteLine( taxi.Disp() & taxi.DispFare() )
```

【実行結果（一部)】

```
燃料：20　速度：60　料金：500
```

『 taxi.Disp() 』は Taxi クラスのオブジェクト内の Disp()メソッドを呼び出していますが、実際は Car クラス（基本クラス）に定義されている Disp()メソッドを呼び出しています。

オブジェクト指向の考え方としては、『 taxi.Disp() 』という記述では、Taxi クラスとしての情報を表示するメソッドであるべきです。そこで、Taxi クラス内に Disp()メソッドを用意することで、『 taxi.Disp() 』の結果を変えてしまおうということです。

書式

```
Class 基本クラス名
    アクセス修飾子 Overridable Sub メソッド名（引数）As 戻り値の型
        ・・・
    End Sub
End Class
```

```
Class 派生クラス名
    Inherits 基本クラス名
    アクセス修飾子 Overrides Sub メソッド名（引数）As 戻り値の型
        ・・・
    End Sub
End Class
```

上記は Sub プロシージャで記述されていますが、Function プロシージャでも同様です。

まず、下記の三点が同じである必要があります。

- メソッド名
- 戻り値の型
- 引数の型、数、順番（引数の変数名は不問）

これらをまとめて、「シグネチャ」と言います。

そして、アクセス修飾子も同じである必要があります。

実装 Sample06

【Car.vb】基本クラス

```
1.  Public Class Car
2.
3.      Public Property Fuel As Integer
4.      Public Property Speed As Integer
5.
6.      Public Sub SetData(tmpFuel As Integer, tmpSpeed As Integer)
7.
8.          Fuel = tmpFuel
9.          speed = tmpSpeed
10.
11.     End Sub
12.
13.     Public Overridable Function Disp() As String
14.
15.         Return "  燃料：" & Fuel & "  速度：" & Speed
16.
17.     End Function
18.
19. End Class
```

【Taxi.vb】派生クラス

```
1.  Public Class Taxi
2.      Inherits Car
3.
4.      Public Property Fare As Integer
5.
6.      Public Overrides Function Disp() As String
7.
8.          Return "  燃料：" & Fuel & "  速度：" & Speed & "  料金：" & Fare
9.
10.     End Function
11.
12. End Class
```

【Module1.vb】

```
1.  Module Module1
2.
3.      Sub Main()
4.
5.          Dim car As Car = New Car()
6.
7.          car.SetData(15, 100)
8.          Console.WriteLine( car.Disp() )
9.
10.         Dim taxi As Taxi = New Taxi()
11.
12.         taxi.SetData(20, 60)
13.         taxi.Fare = 500
14.
15.         Console.WriteLine( taxi.Disp() )
16.
17.     End Sub
18.
19. End Module
```

【実行結果】変更前と同じです。

```
燃料：15  速度：100
燃料：20  速度：60  料金：500
```

解説

下記はオーバーライドの関係になっている部分です。

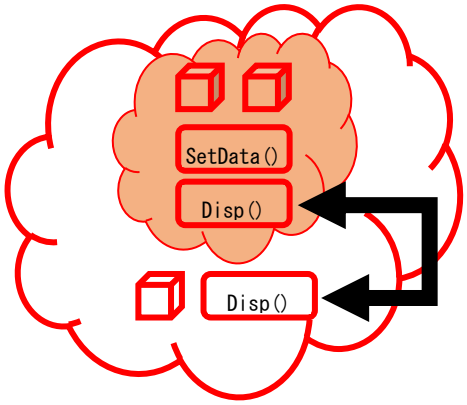
【Car.vb】

```
13.       Public Overridable Function Disp() As String
14.
15.           Return ″  燃料：″ & Fuel & ″  速度：″ & Speed
16.
17.       End Function
```

【Taxi.vb】

```
6.        Public Overrides Function Disp() As String
7.
8.            Return ″  燃料：″ & Fuel & ″  速度：″ & Speed & ″  料金：″ & Fare
9.
10.       End Function
```

下図のように、Taxi クラスのオブジェクト内に 2 つの Disp()メソッドが存在するようになります。ひとつのクラス内に同じシグネチャを持つメソッドが複数存在している場合はエラーになりますが、基本クラスと派生クラスにそれぞれ存在しているので、エラーにはなりません。



このオブジェクトの Disp()メソッドが実行命令を受けた場合、Taxi クラス（派生クラス）に記述された Disp()メソッドが実行され、Car クラス（基本クラス）に記述された Disp()は無視されます。両方実行されるわけではありません。

MyBase（基本クラスのオブジェクトの参照）

派生クラスの Disp()メソッドですが、燃料と速度の表示は基本クラスにも記述されているので、少々無駄に感じます。

派生クラスにてオーバーライドした基本クラスのメソッドは利用できないのでしょうか？Taxi クラスのオブジェクトとして Disp()メソッドを呼び出すと、派生クラスの方の Disp()メソッドが実行されるのは前述のとおりです。

しかし、派生クラス内からであれば、基本クラスの Disp()メソッドは呼び出せます。そのためには、自オブジェクトの内部にある親クラスのオブジェクトを指し示す必要があります。その時に必要なキーワードが「MyBase」です。

下記のように記述を変更し、実行してください。

【Taxi.vb】派生クラス

```
1.  Public Class Taxi
2.      Inherits Car
3.
4.      Public Property Fare As Integer
5.
6.      Public Overrides Function Disp() As String
7.
8.          Return MyBase.Disp() & "  料金：" & Fare
9.
10.     End Function
11.
12. End Class
```

実行結果は先ほどのサンプルと同じです。

```
8.          Return MyBase.Disp() & "  料金：" & Fare
```

網掛け部分の記述が、基本クラスの Disp()メソッドを呼び出しています。MyBase が基本クラスのオブジェクトの部分を指し示すキーワードです。

練習問題 Practice06

先に作成した Rectangle クラスを継承して Triangle クラスを作成します。Triangle クラスの詳細は次のとおりです。ただし、下記の内容は継承を考慮せず、Triangle クラスで必要なすべての情報を記述しています。

コーディングする際は、下記の Rectangle クラスを利用して簡潔に表現しましょう。必要があれば Rectangle クラスを変更してください。

【Rectangle.vb】

```
Public Class Rectangle

    Public Property Width As Integer
    Public Property Height As Integer

    Public Function GetArea() As Double

        Return Width * Height

    End Function

End Class
```

また、作成した Triangle クラスの動作を確認するための main()メソッドは下記を参考にしてください。

【Module1.vb】

```
Module Module1

    Sub Main()

        Triangle クラスのオブジェクトを生成する

        Console.Write("三角形の底辺を入力してください。")

        入力された数字を底辺に設定する

        Console.Write("三角形の高さを入力してください。")

        入力された数字を高さに設定する

        面積を算出し、表示させる

    End Sub

End Module
```

クラス名	Triangle	アクセス修飾子	Public

フィールド

プロパティ名	型	内容
Width	Integer 型	底辺の情報を管理するためのプロパティ
Height	Integer 型	高さの情報を管理するためのプロパティ

コンストラクタ

なし

メソッド

プロシージャ	Function	アクセス修飾子	Public
メソッド名	GetArea	戻り値の型	Double 型
引数	（なし）		

面積を求めて返す

【実行結果例】

```
三角形の底辺を入力してください。8
三角形の高さを入力してください。5
三角形の面積は 20
```

練習問題 解答例 Practice06

【Module1.vb】

```
Module Module1

    Sub Main()

        'Triangle クラスのオブジェクトを生成する
        Dim tri As Triangle = New Triangle()

        Console.Write("三角形の底辺を入力してください。")

        '入力された数字を底辺に設定する
        tri.Width = Integer.Parse(Console.ReadLine())

        Console.Write("三角形の高さを入力してください。")

        '入力された数字を高さに設定する
        tri.Height = Integer.Parse(Console.ReadLine())

        '面積を算出し、表示させる
        Console.WriteLine("三角形の面積は" & tri.GetArea())

End Module
```

【Rectangle.vb】

```
Public Class Rectangle

    Public Property Width As Integer
    Public Property Height As Integer

    Public Overridable Function GetArea() As Double

        Return Width * Height

    End Function

End Class
```

【Triangle.vb】

```
Public Class Triangle
    Inherits Rectangle

    Public Overrides Function GetArea() As Double

        Return MyBase.GetArea / 2

    End Function

End Class
```

ファイル処理

今まで作成したアプリケーションは、その場で実行して終了していました。学習のためのサンプルプログラムはそれでもよいですが、一般的なシステムやツールは、アプリケーションを再起動しても、今までの作業の続きを行いたいはずです。それには、必要な情報をファイルに保存しておき、再び起動した時に読込むことで、前回終了したときを再現する方法が一般的です。

VB にはファイルに対する様々な処理を行うメソッドを持つクラスがあります。それを利用することで、比較的簡単にファイルを扱うことができます。

今回は単純に、指定した文字列をファイルに保存することと、保存したファイルを読込んで画面に出力することを可能とするサンプルの確認をします。

名前空間（namespace）

ファイルへ書き込むには StreamWriter クラスを、ファイルから読み込むには StreamReader クラスを利用します。ただし、これらのクラスは標準的に利用できるクラスではありません。そのため、そのクラスがどこに存在しているかを記述する必要があります。そこで指定するのが名前空間です。

```
Imports 名前空間名
```

上記のように記述すると、指定した名前空間に含まれるクラスなどが利用可能になります。

StreamWriter クラスと StreamReader クラスは System.IO 名前空間のクラスですので、下記のような記述が必要です。

```
1.    Imports System.IO
```

名前空間はフォルダのような階層構造になっており、System というグループの中の IO というグループ内の機能を利用可能にしています。

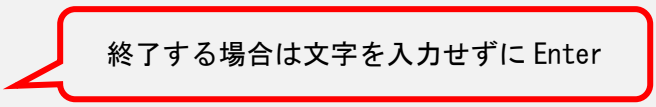
このように、提供されているさまざまなクラスは名前空間というグループに所属しており、標準的に利用可能なもの以外の機能のクラスを利用する場合は Imports 宣言が必要になります。

実装 Sample07

```
1.  Imports System.IO
2.
3.  Module Module1
4.
5.      Dim filename As String
6.
7.      Sub Main()
8.
9.          Console.Write("ファイル名を指定してください  ")
10.         filename = Console.ReadLine()
11.
12.         Do
13.             Console.Write("1:書込  2:読込  0:終了  ")
14.             Select Case Integer.Parse(Console.ReadLine())
15.                 Case 1 : WriteText()
16.                 Case 2 : ReadText()
17.                 Case Else : Exit Do
18.             End Select
19.         Loop
20.
21.     End Sub
22.
23.     Private Sub WriteText()
24.
25.         Dim sw As New StreamWriter("C:¥workspace¥" & filename)
26.         Dim str As String
27.
28.         Console.WriteLine("書き込む文字列を入力してください")
29.         Console.WriteLine("空文字を指定すると終了します")
30.
31.         Do
32.             str = Console.ReadLine()
33.             If str = "" Then Exit Do
34.             sw.WriteLine(str)
35.         Loop
36.
37.         sw.Close()
38.
39.     End Sub
40.
41.     Private Sub ReadText()
42.
43.         Dim sr As New StreamReader("C:¥workspace¥" & filename)
44.         Dim str As String = sr.ReadToEnd()
45.
46.         Console.WriteLine(str)
47.
48.         sr.Close()
49.
50.     End Sub
51.
52. End Module
```

【実行結果例】

```
ファイル名を指定してください  test.txt
1:書込  2:読込  0:終了  1
書き込む文字列を入力してください
空文字を指定すると終了します
abcdefg
hijklmn
opqrstu
vwxyz

1:書込  2:読込  0:終了  2
abcdefg
hijklmn
opqrstu
vwxyz

1:書込  2:読込  0:終了  0
```



解説

```
1.     Imports System.IO
```

StreamWriter クラスおよび StreamReader クラスは System.IO 名前空間のクラスなので、Imports 宣言が必要です。

```
5.         Dim filename As String
6.
7.         Sub Main()
8.
9.             Console.Write("ファイル名を指定してください  ")
10.            filename = Console.ReadLine()
11.
12.            Do
13.                Console.Write("1:書込  2:読込  0:終了  ")
14.                Select Case Integer.Parse(Console.ReadLine())
15.                    Case 1 : WriteText()
16.                    Case 2 : ReadText()
17.                    Case Else : Exit Do
18.                End Select
19.            Loop
20.
21.        End Sub
```

Main()メソッドは基本的な文法しか扱っていませんが、コンソールアプリケーションの対話を実現するための基本的な形式です。コンソールへ入力を促すメッセージを表示し、入力された内容を取得し、その入力内容によって決まった処理をすることを繰り返します。

VB では、: コロンを使用すると、通常複数行で書かなければいけない内容を改行せずに記述できます。

通常は 1 命令を 1 行（長い場合は複数行）で書くのが基本ですが、下記左側のように Case ごとの処理が短い場合、行が増えるうえにインデント（タブでの字下げ）も深くなってしまいます。可読性の面で、下記右側のような記述を選択することがあります。

```
Select Case Integer.Parse(Console.ReadLine())
    Case 1
        WriteText()
    Case 2
        ReadText()
    Case Else
        Exit Do
End Select
```

```
Select Case Integer.Parse(Console.ReadLine())
    Case 1 : WriteText()
    Case 2 : ReadText()
    Case Else : Exit Do
End Select
```

下記は書込み用のプロシージャの部分です。

```
23.       P rivate Sub WriteText()
24.
25.           Dim sw As New StreamWriter("C:¥workspace¥" & filename)
26.           Dim str As String
27.
28.           Console.WriteLine("書き込む文字列を入力してください")
29.           Console.WriteLine("空文字を指定すると終了します")
30.
31.           Do
32.               str = Console.ReadLine()
33.               If str = "" Then Exit Do
34.               sw.WriteLine(str)
35.           Loop
36.
37.           sw.Close()
38.
39.       End Sub
```

WriteText()メソッドはテキストデータをファイルへ書込むためのメソッドです。ファイルの書込み方法は何種類もありますが、今回はとても単純で扱いやすい StreamWriter クラスを利用しています。

StreamWriter クラスのコンストラクタは、受け取ったファイル名のファイルを開いて準備します。存在しない場合は自動的に新規作成します。

StreamWriter クラスの WriteLine()メソッドは、コンストラクタで指定したファイルに対して文字列を書込むメソッドです。

書込みが終わったら Close()メソッドでファイルを閉じます。

下記は読込み用のプロシージャの部分です。

```
41.       Private Sub ReadText()
42.
43.           Dim sr As New StreamReader("C:¥workspace¥" & filename)
44.           Dim str As String = sr.ReadToEnd()
45.
46.           Console.WriteLine(str)
47.
48.           sr.Close()
49.
50.       End Sub
```

StreamReader クラスは書込み用の StreamWriter クラスと対になるクラスで、テキストデータをファイルから読込むためのクラスです。コンストラクタへ指定した、ファイルの内容を取得できます。ReadToEnd()メソッドはファイルの内容のすべてを一度に読込むメソッドです。ファイルの内容が改行を含んでいても、改行を含めて String 型のデータとして取得できます。

文字コード

コンピュータの中ではすべてのデータは数値（究極的には 0 と 1 のビット情報）で管理されています。文字データも内部では文字コードという数値で管理されています。（下表参照）

画面上に表示される文字はもちろん、「空白（スペース）」や「インデント（タブ）」のように画面上には何も表示されていないものもコードとしては存在しています。そのようなコードを制御コードと言います。

「改行」も同様です。「改行」は特殊で、2 つのコードで成り立っています。次の文字を表示させる位置を『（同じ横位置の）次の行へ縦移動させるコード』と『（同じ行の）先頭へ横移動させるコード』の組み合わせです。

メモリ内の文字コードの状態が下図左のような状態の時、その部分を文字列として表示させた場合は下図右のようになります。

41	21	0D	0A	62	20	3F	00

```
A!
b ?
```

20 はスペース、00 は文字の終端を表します

文字コード	表示内容	文字コード	表示内容	文字コード	表示内容	文字コード	表示内容	文字コード	表示内容	文字コード	表示内容	文字コード	表示内容	文字コード	表示内容
00	NUL	10	DLE	20	SP	30	0	40	@	50	P	60	`	70	p
01	SOH	11	DC1	21	!	31	1	41	A	51	Q	61	a	71	q
02	STX	12	DC2	22	″	32	2	42	B	52	R	62	b	72	r
03	ETX	13	DC3	23	#	33	3	43	C	53	S	63	c	73	s
04	EOT	14	DC4	24	$	34	4	44	D	54	T	64	d	74	t
05	ENQ	15	NAK	25	%	35	5	45	E	55	U	65	e	75	u
06	ACK	16	SYN	26	&	36	6	46	F	56	V	66	f	76	v
07	BEL	17	ETB	27	’	37	7	47	G	57	W	67	g	77	w
08	BS	18	CAN	28	(	38	8	48	H	58	X	68	h	78	x
09	HT	19	EM	29	)	39	9	49	I	59	Y	69	i	79	y
0A	LF	1A	SUB	2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
0B	VT	1B	ESC	2B	+	3B	;	4B	K	5B	[	6B	k	7B	{
0C	FF	1C	FS	2C	,	3C	<	4C	L	5C	¥	6C	l	7C	|
0D	CR	1D	GS	2D	-	3D	=	4D	M	5D	]	6D	m	7D	}
0E	SO	1E	RS	2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
0F	SI	1F	US	2F	/	3F	?	4F	O	5F	_	6F	o	7F	DEL

濃い網掛け部分は制御コードです。

練習問題 Practice07

下記のような 2 つのプロジェクトを作成してください。

プロジェクト	Practice07w
モジュール	Module1
プロシージャ	Main

処理
実行時の日時をワークスペースの直下に「実行日時.txt」という名前で保存してください。

【実行結果】

```
実行日時を「実行日時.txt」に保存しました
```

プロジェクト	Practice07r
モジュール	Module1
プロシージャ	Main

処理
ワークスペースの直下の「実行日時.txt」に記載されている文字列を読み取り、コンソールへ表示させて下さい。

【実行結果例】

```
「実行日時.txt」に保存されている日付は「2000/11/11 12:34:56」です
```

表示される日時は「実行日時.txt」に保存されている日時であること。

現在の日時

実行時の日時は DateAndTime クラスの Now プロパティで取得可能です。具体的には、下記のように使用します。

Now プロパティは Date 型のデータとして扱えます

```
Dim exetime As Date = DateAndTime.Now
Console.WriteLine( exetime )
```

変数に格納せず、直接扱うこともできます

```
Console.WriteLine( DateAndTime.Now )
```

クラス名の部分は省略可能です

```
Console.WriteLine( Now )
```

練習問題 解答例 **Practice07**

【プロジェクト：Practice07w　Module1.vb】

```
1.  Imports System.IO
2.
3.  Module Module1
4.
5.      Sub Main()
6.
7.          Dim sw As New StreamWriter("C:¥workspace¥実行日時.txt")
8.
9.          sw.Write( DateAndTime.Now )
10.
11.         sw.Close()
12.
13.         Console.WriteLine("実行日時を「実行日時.txt」に保存しました")
14.
15.     End Sub
16.
17. End Module
```

【プロジェクト：Practice07r　Module1.vb】

```
1.  Imports System.IO
2.
3.  Module Module1
4.
5.      Sub Main()
6.
7.          Dim sr As New StreamReader("C:¥workspace¥実行日時.txt")
8.
9.          Console.Write("「実行日時.txt」に保存されている日付は「")
10.
11.         Console.WriteLine(sr.ReadToEnd() & "」です")
12.
13.         sr.Close()
14.
15.     End Sub
16.
17. End Module
```

例外処理

例外とは、プログラム実行中に発生するエラーのことです。今までは「エラー」と表現してきましたが、実際は「例外」と言うのが正しい表現です。エラーというのは、OS やメモリなどの要因でプログラムの続行に影響が出る場合であることが多いです。言語によって線引きが変わるので、厳密に分けることはできませんが、プログラムの続行に際して致命的なものはエラー、そうでないものは例外と思ってください。

まずは、わざと例外を発生させてみましょう。次のサンプルを実行してください。

```
1.  Module Module1
2.
3.      Sub Main()
4.
5.          Console.Write("数字を入力してください。")
6.          Dim num As Integer = Integer.Parse( Console.ReadLine() )
7.          Console.WriteLine("入力された数字は" & num)
8.
9.      End Sub
10.
11. End Module
```

実行時、数値変換できる文字を入力した場合は問題ありませんが、アルファベットなどの数値変換できない文字を入力した場合、次のようなメッセージが表示されます。

```
数字を入力してください。ABC

ハンドルされていない例外: System.FormatException: 入力文字列の形式が正しくありません。
   場所 System.Number.StringToNumber(String str, NumberStyles options, NumberBuffer& number, N
umberFormatInfo info, Boolean parseDecimal)
   場所 System.Number.ParseInt32(String s, NumberStyles style, NumberFormatInfo info)
   場所 System.Int32.Parse(String s)
   場所 ConsoleApp4.Module1.Main() 場所 C:¥workspace¥ConsoleApp1¥ConsoleApp4¥Module1.vb:行 6
```

また、"入力された数字は～"という表示がされていないことから、プログラムが中断されてしまったことが分かります。

コンソールに表示されたメッセージの一行目は、「System.FormatException」という種類の例外が発生し、その例外への対処がされていないことを示しています。『System』は名前空間を意味し、『FormatException』はクラス名を意味します。

VB は実行時のエラーの情報を管理することにもオブジェクトを活用しています。オブジェクトを利用しているということは、クラスが存在しているということです。例外の種類ごとにクラスが定義されており、基本的に〇〇Exception という名前がついています。そして、そのすべての例外のクラスは System.Exception クラスを基本クラスとしています。

今回の『FormatException』は Parse()メソッドのような、変換する処理がうまくいかなかったときに発生する例外です。

処理に使用する値を事前にチェックすれば、例外は発生しませんが、扱うデータの量やパターンが多い場合は、すべてのチェックを行うのは大変ですし、漏れが生じる可能性があります。

例外処理の機構がなかった時代のプログラミング言語では、値の妥当性をチェックしてから処理をするしかありませんでした。しかし、例外処理の機構をもつ、現代のプログラミング言語では、値の妥当性をチェックせず、行いたい処理を記述し、もしもその過程で例外が発生した時はプログラムを中断させずに処理を継続させる形式をとっています。

書式

```
Try
        例外が発生する可能性がある処理          } ①
        ・・・
Catch  変数名  As  例外クラス名
        上記例外クラス名の例外が発生した時の処理  } ②
        ・・・
Finally
        例外が発生しても発生しなくても実行する処理  } ③
        ・・・
End  Try
```

①で例外が発生しない場合、下記のように実行される

（前処理） → ① → ③ → （後処理）

①で例外が発生し、Catch される場合、下記のように実行される

（前処理） → ① → （例外発生） → ② → ③ → （後処理）

①で例外が発生し、Catch されない場合、下記のように実行される

（前処理） → ① → （例外発生） → ③ → 【処理終了（中断）】

実装 Sample08

```
1.  Module Module1
2.
3.      Sub Main()
4.
5.          Console.Write("数字を入力してください。")
6.          Try
7.
8.              Dim num As Integer = Integer.Parse( Console.ReadLine() )
9.              Console.WriteLine("入力された数字は" & num)
10.
11.         Catch ex As FormatException
12.
13.             Console.WriteLine("不正な入力です")
14.             Console.WriteLine( ex.ToString() )
15.
16.         Finally
17.
18.             Console.WriteLine("処理の終了準備をします")
19.
20.         End Try
21.
22.         Console.WriteLine("処理を終了します")
23.
24.     End Sub
25.
26. End Module
```

【実行結果例 1】

```
数字を入力してください。100
入力された数字は 100
処理の終了準備をします
処理を終了します
```

【実行結果例 2】

```
数字を入力してください。ABC
不正な入力です
System.FormatException: 入力文字列の形式が正しくありません。
   場所 System.Number.StringToNumber(String str, NumberStyles options, Numb
erBuffer& number, NumberFormatInfo info, Boolean parseDecimal)
   場所 System.Number.ParseInt32(String s, NumberStyles style, NumberFormat
Info info)
   場所 System.Int32.Parse(String s)
   場所 ConsoleApp4.Module1.Main() 場所 C:¥workspace¥ConsoleApp1¥ConsoleApp
4¥Module1.vb:行 8
処理の終了準備をします
処理を終了します
```

【実行結果例 3】

```
数字を入力してください。12345678910

ハンドルされていない例外: System.OverflowException: Int32 型の値が大きすぎるか、または小さすぎます。
   場所 System.Number.ParseInt32(String s, NumberStyles style, NumberFormatInfo info)
   場所 System.Int32.Parse(String s)
   場所 ConsoleApp4.Module1.Main() 場所 C:¥workspace¥ConsoleApp1¥ConsoleApp4¥Module1.vb:行 8
処理の終了準備をします
```

解説

```
6.          Try
7.
8.              Dim num As Integer = Integer.Parse( Console.ReadLine() )
9.              Console.WriteLine("入力された数字は" & num)
10.
11.         Catch ex As FormatException
12.
13.             Console.WriteLine("不正な入力です")
14.             Console.WriteLine( ex.ToString() )
15.
16.         Finally
17.
18.             Console.WriteLine("処理の終了準備をします")
19.
20.         End Try
```

Try～Catch までの間で例外が発生した場合は Catch の部分へ行きます。今回は FormatException クラスの例外が発生した場合に備えて 11 行目で定義しています。発生した例外オブジェクトを ex という変数で扱えるようになっているので、ToString()メソッドで例外の情報を表示させています。これは学習用に出力しているだけですので、通常のアプリケーションでは表示することはないでしょう。

Finally～End Try の部分は Try～Catch の間で例外が発生しても発生しなくても実行される部分です。

練習問題 Practice08

先に確認したファイル処理のサンプルは、ファイル読込み時に存在しないファイル名を指定した場合やファイル書込み時にファイル名として指定できない文字が含まれる場合など、例外が発生して処理が終了してしまいます。例外処理を施すことで、安全に処理が継続できるように変更してください。

今回は例外発生の原因の詳細は区別しません。

【実行結果例】

```
ファイル名を指定してください　test?txt
1:書込　2:読込　0:終了　1
ファイル操作のエラーです

ファイル名を指定してください　test.txt
1:書込　2:読込　0:終了　2
ファイル操作のエラーです

ファイル名を指定してください　abc
1:書込　2:読込　0:終了　0
```

? はファイル名に使用できません

存在しないファイルを指定する

終了選択時は全体を終了させる

test.txt が存在する場合は削除をしてから実行してください。

練習問題 解答例 **Practice08**

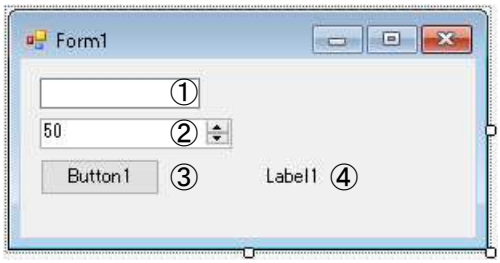
```
Imports System.IO

Module Module1

    Dim filename As String

    Sub Main()

        Do
            Try
                Console.Write("ファイル名を指定してください  ")
                filename = Console.ReadLine()

                Do
                    Console.Write("1:書込  2:読込  0:終了  ")
                    Select Case Console.ReadLine()
                        Case 1 : WriteText()
                        Case 2 : ReadText()
                        Case Else : Exit Sub
                    End Select
                Loop

            Catch ex As Exception
                Console.WriteLine("ファイル操作のエラーです")
                Console.WriteLine()
            End Try
        Loop

    End Sub

    Private Sub WriteText()

                                    (省略)

    End Sub

    Private Sub ReadText()

                                    (省略)

    End Sub

End Module
```


課題（クラスとオブジェクト）

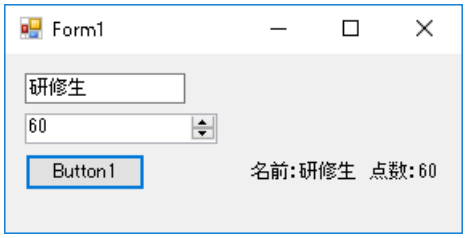
フォームアプリケーションにおいてもクラスとオブジェクトの振る舞いは変わりません。クラスとオブジェクトを扱うための基本的な文法を確認しながらフォームアプリケーションに組み込んでみましょう。

サンプル１

【デザイン例】



【実行結果例】



番号	コントロール名	プロパティ名	プロパティ値
①	TextBox	―	―
②	NumericUpDown	Increment	5
		Maximum	100
		Minimum	0
		Value	50
③	Button	―	―
④	Label	―	―

コントロール	③	イベント	Click
Student クラスのオブジェクトを生成する。 そのオブジェクトの SetData()メソッドを利用して①と②に入力された値を、そのオブジェクトへ設定する。 そのオブジェクトの Disp()メソッドの結果を④へ表示させる。			

クラス名	Student

フィールド

変数名	型	内容
name	String 型	名前を管理するための変数
score	Integer 型	点数を管理するための変数

メソッド

プロシージャ	Sub		
メソッド名	SetData	戻り値の型	（なし）
引数	String 型, Integer 型		
第一引数で受け取った値をフィールドプロパティ name へ格納する。 第二引数で受け取った値をフィールドプロパティ score へ格納する。			
プロシージャ	Function		
メソッド名	Disp	戻り値の型	String 型
引数	（なし）		
フィールドプロパティ name および score をもとに表示文字列を返す。 表示例：「　名前：研修生　点数：60　」			

NumericUpDown

数字を選択させるためのコントロールです。テキストボックスのように直接入力もできますが、上下のボタンを使って値を増減できます。主要なプロパティは下表のとおりです。

プロパティ	設定内容
DecimalPlaces	表示する小数点以下の桁数
Minimum	コントロールで設定可能な下限値
Maximum	コントロールで設定可能な上限値
Increment	上下ボタンを操作した時の増減値
Value	現在の値を設定および取得できる

実装 ExerciseSample01

【Form1.vb】

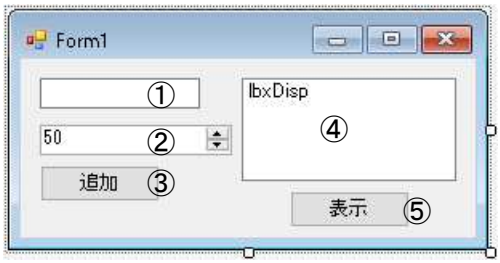
```
Public Class Form1

    Private Sub Button1_Click(sender As Object,
                              e As EventArgs) Handles Button1.Click

        Dim std As Student = New Student()
        std.SetData(TextBox1.Text, NumericUpDown1.Value)

        Label1.Text = std.Disp()

    End Sub

End Class
```

【Student.vb】

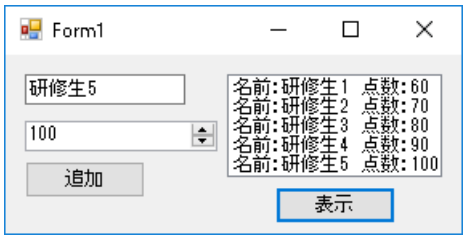
```
Public Class Student

    Dim name As String
    Dim score As Integer

    Sub SetData(tmpName As String, tmpScore As Integer)

        name = tmpName
        score = tmpScore

    End Sub

    Function Disp() As String

        Return "名前：" & name & "  点数：" & score

    End Function

End Class
```

課題 1

【デザイン例】



【実行結果例】



番号	コントロール名	プロパティ名	プロパティ値
①	TextBox	(Name)	txtName
②	NumericUpDown	(Name)	nudScore
		Increment	5
		Maximum	100
		Minimum	0
		Value	50
③	Button	(Name)	btnAdd
		Text	追加
④	ListBox	(Name)	lbxDisp
⑤	Button	(Name)	btnDisp
		Text	表示

フィールド		
students	Student 型の配列	Student のオブジェクトの参照を格納するための配列。要素数は 5 とする（要素番号は 0～4）
idx	Integer 型	上記の配列の要素番号を管理するための変数

コントロール	③	イベント	Click
Student クラスのオブジェクトを生成する。 そのオブジェクトの SetData()メソッドを利用して①と②に入力された値を、そのオブジェクトへ設定する。 配列 students の要素番号 idx の位置へそのオブジェクトを格納する。 （このボタンが押されるたびにオブジェクトを生成して配列へ格納する。5 つのオブジェクトを格納したあとはメッセージを表示してオブジェクトを生成させない。）			
コントロール	⑤	イベント	Click
配列 students 内のそれぞれのオブジェクトの Disp()メソッドの結果を④へ表示させる。			

クラス名	Student

フィールド

変数名	型	内容
name	String 型	名前を管理するための変数
score	Integer 型	点数を管理するための変数

メソッド

プロシージャ	Sub		
メソッド名	SetData	戻り値の型	（なし）
引数	String 型, Integer 型		
第一引数で受け取った値をフィールドプロパティ name へ格納する。 第二引数で受け取った値をフィールドプロパティ score へ格納する。			
プロシージャ	Function		
メソッド名	Disp	戻り値の型	String 型
引数	（なし）		
フィールドプロパティ name および score をもとに表示文字列を返す。 表示例：「　名前：研修生　点数：60　」			

ListBox の更新

ListBox に対して内容を追加する方法は下記のとおりです。

```
ListBox.Items.Add　追加する内容
```

今回はプログラムで 5 件以上は追加されないようにしていますが、ListBox 自体は何件でも追加できます。フォーム上に表示しきれない場合は ListBox の右側にスクロールバーが表示されます。

オブジェクトの配列

変数には値型と参照型があります。Integer 型に代表される、値を直接扱う値型に対して、オブジェクトの参照を扱う変数が参照型です。どちらの変数も直接的か間接的かの違いはありますが、何かデータを扱っていることには変わりません。

配列も同様です。値型の変数をいくつかまとめて扱う配列があれば、参照型の変数をいくつかまとめて扱う配列もあります。

基本的な使い方は同じです。配列の内の一つを扱いたい場合は、要素番号（インデックス）を指定します。要素番号（インデックス）は 0 から始まる点も同じです。

解答例 Exercise01

【Student.vb】

```
Public Class Student

    Dim name As String
    Dim score As Integer

    Sub SetData(tmpName As String, tmpScore As Integer)

        name = tmpName
        score = tmpScore

    End Sub

    Function Disp() As String

        Return "名前：" & name & "　点数：" & score

    End Function

End Class
```

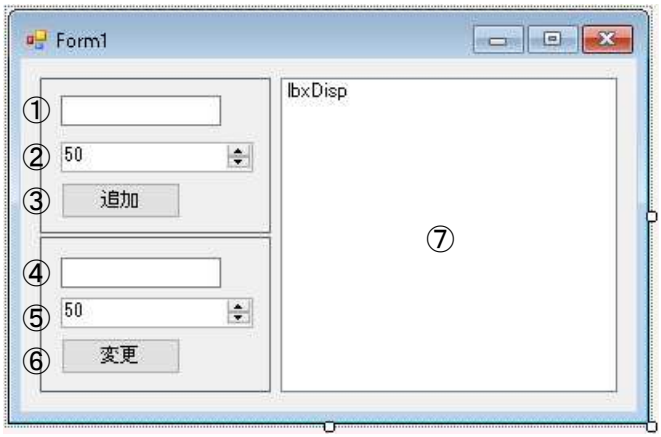
【Form1.vb】

```
Public Class Form1

    Dim students(4) As Student
    Dim idx As Integer = 0

    Private Sub btnAdd_Click(sender As Object,
                             e As EventArgs) Handles btnAdd.Click

        If idx <= 4 Then

            Dim std As Student = New Student()
            std.SetData(txtName.Text, nudScore.Value)

            students(idx) = std
            idx += 1

        Else
            MsgBox("すでに5名登録済みです")
        End If

    End Sub

    Private Sub btnDisp_Click(sender As Object,
                              e As EventArgs) Handles btnDisp.Click

        For cnt As Integer = 0 To 4

            Dim std As Student = students(cnt)
            Dim str As String = std.Disp()
            lbxDisp.Items.Add(str)

        Next

    End Sub

End Class
```

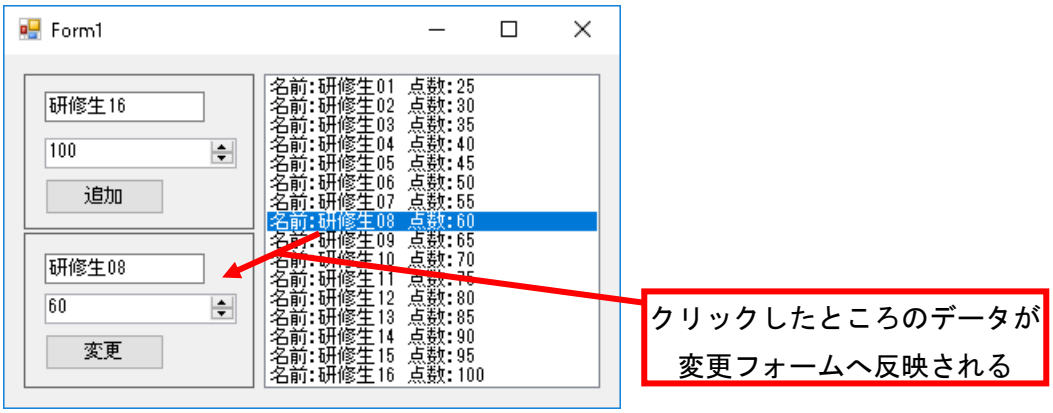
課題（コンストラクタ）

サンプル 2

【デザイン例】



【実行結果例】



番号	コントロール名	プロパティ名	プロパティ値
①	TextBox	(Name)	txtAddName
②	NumericUpDown	(Name)	nudAddScore
		Increment	5
		Value	50
③	Button	(Name)	btnAdd
		Text	追加
④	TextBox	(Name)	txtUpdName
⑤	NumericUpDown	(Name)	nudUpdScore
		Increment	5
		Value	50

⑥	Button	(Name)	btnUpd
		Text	変更
⑦	ListBox	(Name)	lbxDisp

フィールド		
students	Student 型の配列	Student のオブジェクトの参照を格納するための配列。要素数は 16 とする（要素番号は 0～15）
idx	Integer 型	上記の配列の要素番号を管理するための変数

コントロール	③	イベント	Click
Student クラスのオブジェクトを生成する。 その際、コンストラクタを利用して①と②に入力された値を、そのオブジェクトへ設定する。 そのオブジェクトの Disp()メソッドの結果を⑦へ表示させる。 ただし、16 件を超える場合はメッセージを表示してオブジェクトを生成させない。			
コントロール	⑥	イベント	Click
⑦の選択位置を取得し、選択されていない場合はイベントプロシージャを終了する。 フィールド変数 students 配列内から、⑦の選択位置のインデックスを持つ情報を取得する。 そのオブジェクトの SetData()メソッドを利用して、④と⑤に入力された内容を、そのオブジェクトへ設定する。 そのオブジェクトの Disp()メソッドを利用して⑦の選択位置の内容を更新する。			
コントロール	⑦	イベント	SelectedIndexChanged
フィールド変数 students 配列内から、⑦の選択位置のインデックスを持つ情報を取得する。 そのオブジェクトの GetName()メソッドの結果を④へ表示させる。 そのオブジェクトの GetScore()メソッドの結果を⑤へ表示させる。			

⑦のイベントは動作上必須ではありませんが、登録内容を変更したいときに変更前の現在の状況が入力フォームへ反映されたほうが便利です。フォームアプリケーションは、使う人が使いやすいように考えて作る必要があります。

クラス名	Student

フィールド

変数名	型	内容
name	String 型	名前を管理するための変数
score	Integer 型	点数を管理するための変数

コンストラクタ

プロシージャ	Sub
引数	String 型, Integer 型
第一引数で受け取った値をフィールドプロパティ name へ格納する。 第二引数で受け取った値をフィールドプロパティ score へ格納する。	

メソッド

プロシージャ	Sub		
メソッド名	SetData	戻り値の型	（なし）
引数	String 型, Integer 型		
第一引数で受け取った値をフィールドプロパティ name へ格納する。 第二引数で受け取った値をフィールドプロパティ score へ格納する。			
プロシージャ	Function		
メソッド名	Disp	戻り値の型	String 型
引数	（なし）		
フィールドプロパティ name および score をもとに表示文字列を返す。 表示例：「　名前：研修生　点数：60　」			
プロシージャ	Function		
メソッド名	GetName	戻り値の型	String 型
引数	（なし）		
フィールドプロパティ name を返す。			
プロシージャ	Function		
メソッド名	GetScore	戻り値の型	Integer 型
引数	（なし）		
フィールドプロパティ score を返す。			

ListBox の更新

ListBox に対して内容を追加する場合は「ListBox.Items.Add(追加する内容)」のように記述しましたが、更新する方法は下記のとおりです。

```
ListBox.Items( index ) = 更新する内容
```

メソッドではなく、直接値を代入することで更新が可能です。更新するときは既にあるデータのインデックスの指定が必要です。

ListBox 内の選択されている位置を取得する方法は下記のとおりです。

```
Dim idx As Integer = ListBox.SelectedIndex
```

選択されていない場合は-1 が取得され、選択されている場合は 0 以上の値が取得されます。ListBox 内の一番上の要素が選択されている場合は 1 ではなく 0 ですので注意してください。

ListBox のイベント

ListBox をフォームデザイナー上でダブルクリックすると、Click イベントではなく、SelectedIndexChanged イベントが自動生成されます。これは、ListBox 内の要素の選択状況に変化があったとき用のイベントですが、クリックしてもこのイベントは発生します。

ListBox が更新された時、見た目の選択状況は変わりませんが、内部的に、一時的に選択されているインデックスが-1（未選択）の状態になり、SelectedIndexChanged イベントが発生してしまいます。

よって、未選択の状態の場合は何もしないというような回避策が必要です。

Panel

フォーム、をよく見てみると、追加の操作用のコントロールと変更の操作用のコントロールが線で囲まれてグループ化されていることが分かります。Panel というコントロールを利用すると、コントロールをいくつかグループ化して管理できます。座標の移動もまとめて行えます。見た目にも囲みの線が表示されるので、わかりやすくなります。囲みの線は調整可能です。

実装 ExerciseSample02

【Form1.vb】

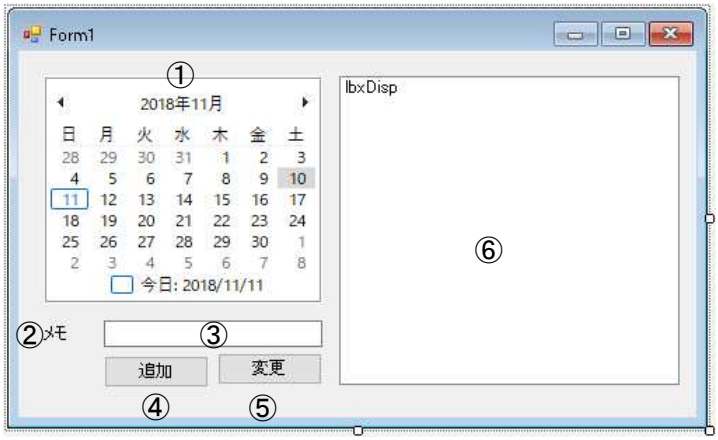
```
Public Class Form1

    Dim students(15) As Student
    Dim idx As Integer = 0

    Private Sub btnAdd_Click(sender As Object,
                             e As EventArgs) Handles btnAdd.Click

        If idx <= 15 Then
            Dim std As Student =
                New Student(txtAddName.Text, nudAddScore.Value)
            students(idx) = std
            idx += 1

            lbxDisp.Items.Add( std.Disp() )
        Else
            MsgBox("すでに 16 名登録済みです")
        End If

    End Sub

    Private Sub btnUpd_Click(sender As Object,
                                e As EventArgs) Handles btnUpd.Click

        Dim idx As Integer = lbxDisp.SelectedIndex

        If idx < 0 Then
            MsgBox("変更対象を選択してください")
            Exit Sub
        End If

        Dim std As Student = students(idx)
        std.SetData(txtUpdName.Text, nudUpdScore.Value)
        lbxDisp.Items(idx) = std.Disp()

    End Sub

    Private Sub lbxDisp_SelectedIndexChanged(sender As Object,
                e As EventArgs) Handles lbxDisp.SelectedIndexChanged

        Dim idx As Integer = lbxDisp.SelectedIndex

        If idx < 0 Then
            Exit Sub
        End If

        Dim std As Student = students(idx)
        txtUpdName.Text = std.GetName()
        nudUpdScore.Value = std.GetScore()

    End Sub

End Class
```


【Student.vb】

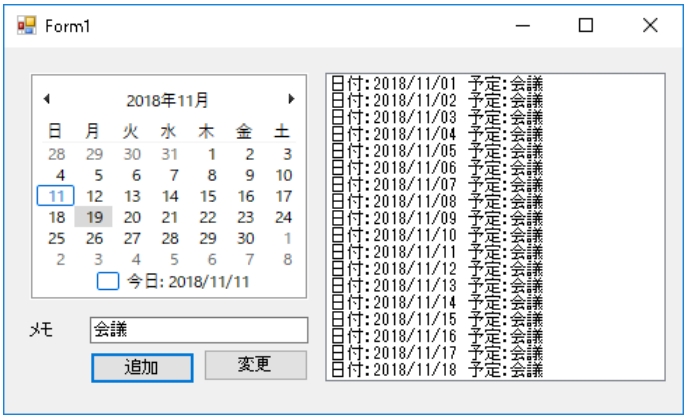
```
Public Class Student

    Dim name As String
    Dim score As Integer

    Sub SetData(tmpName As String, tmpScore As Integer)

        name = tmpName
        score = tmpScore

    End Sub

    Function Disp() As String

        Return "名前：" & name & "  点数：" & score

    End Function

    Sub New(tmpName As String, tmpScore As Integer)

        name = tmpName
        score = tmpScore

    End Sub

    Function GetName() As String

        Return name

    End Function

    Function GetScore() As Integer

        Return score

    End Function

End Class
```

課題 2

【デザイン例】



【実行結果例】



番号	コントロール名	プロパティ名	プロパティ値
①	MonthCalendar	MaxSelectionCount	1
②	Label	Text	メモ
③	TextBox	(Name)	txtMemo
④	Button	(Name)	btnAdd
		Text	追加
⑤	Button	(Name)	btnUpd
		Text	変更
⑥	ListBox	(Name)	lbxDisp

フィールド		
schedules	Schedule 型の配列	Schedule のオブジェクトの参照を格納するための配列。要素数は 18 とする（要素番号は 0～17）
idx	Integer 型	上記の配列の要素番号を管理するためのカウンタ変数

コントロール	④	イベント	Click
Schedule クラスのオブジェクトを生成する。 その際、コンストラクタを利用して①と③に入力された値を、そのオブジェクトへ設定する。 そのオブジェクトの Disp()メソッドの結果を⑥へ表示させる。 ただし、18 件を超える場合はメッセージを表示してオブジェクトを生成させない。			
コントロール	⑤	イベント	Click
⑥の選択位置を取得し、選択されていない場合はイベントプロシージャを終了する。 フィールド変数 schedules 配列内から、⑥の選択位置のインデックスを持つ情報を取得する。 そのオブジェクトの SetData()メソッドを利用して、①と③に入力された内容を、そのオブジェクトへ設定する。 そのオブジェクトの Disp()メソッド利用して⑥の選択位置の内容を更新する。			
コントロール	⑥	イベント	SelectedIndexChanged
⑥の選択位置を取得し、選択されていない場合はイベントプロシージャを終了する。 フィールド変数 schedules 配列内から、⑥の選択位置のインデックスを持つ情報を取得する。 そのオブジェクトの GetSchDate()メソッドを利用して①の SetSelectionRange プロパティを設定する。選択範囲は 1 日でなので、開始位置（第一引数）と終了位置（第二引数）は同じ。 同じく、そのオブジェクトの GetMemo()メソッドを利用して、③の Text プロパティを設定する。			

クラス名	Schedule

フィールド

変数名	型	内容
schDate	Date 型	日付を管理するための変数
memo	String 型	予定を管理するための変数

コンストラクタ

プロシージャ	Sub
引数	Date 型, String 型
第一引数で受け取った値をフィールドプロパティ schDate へ格納する。 第二引数で受け取った値をフィールドプロパティ memo へ格納する。	

メソッド

プロシージャ	Sub		
メソッド名	SetData	戻り値の型	（なし）
引数	Date 型, String 型		
第一引数で受け取った値をフィールドプロパティ schDate へ格納する。 第二引数で受け取った値をフィールドプロパティ memo へ格納する。			
プロシージャ	Function		
メソッド名	DispSchedule	戻り値の型	String 型
引数	（なし）		
フィールドプロパティ schDate および memo をもとに表示文字列を返す。 表示例：「　日付：2000/10/20　予定：会議　」			
プロシージャ	Function		
メソッド名	GetSchDate	戻り値の型	Date 型
引数	（なし）		
フィールドプロパティ schDate を返す。			
プロシージャ	Function		
メソッド名	GetMemo	戻り値の型	String 型
引数	（なし）		
フィールドプロパティ memo を返す。			

MonthCalendar と Date 型

MonthCalendar はカレンダーを表示して、日付を選択させるためのコントロールです。デフォルトでは 7 日間の範囲を選択できますが、今回は MaxSelectionCount プロパティを 1 に設定することで 1 日だけを選択させるようにしています。

日曜日を赤色にするなど、このコントロール上の日付の表示を変化させることは単純にはできません。実現したい内容によっては、日付の数の分だけラベルを配置するなどして自作する必要があるかもしれません。

選択範囲は SelectionStart プロパティと SelectionEnd プロパティで取得できます。今回は 1 日分しか選択できないようにしているのでどちらで取得しても同じです。

取得できるデータの型は日付の形式を持つ Date 型です。Date 型は 1 を足すと翌日を表し、1 を引くと前日を表すデータ型で、日付の計算をするのにとても便利です。月や年をまたぐ計算や閏年を考慮した計算も可能です。

解答例 Exercise02

【Form1.vb】

```
Public Class Form1

    Dim schedules(17) As Schedule
    Dim idx As Integer = 0

    Private Sub btnAdd_Click(sender As Object,
                             e As EventArgs) Handles btnAdd.Click

        If idx <= 17 Then
            Dim sch As Schedule =
                New Schedule(MonthCalendar1.SelectionStart, txtMemo.Text)
            schedules(idx) = sch
            idx += 1

            lbxDisp.Items.Add(sch.DispSchedule())
        Else
            MsgBox("すでに18件登録済みです")
        End If

    End Sub

    Private Sub btnUpd_Click(sender As Object,
                             e As EventArgs) Handles btnUpd.Click

        Dim id As Integer = lbxDisp.SelectedIndex
        If id < 0 Then
            MsgBox("変更対象を選択してください")
            Exit Sub
        End If

        Dim sch As Schedule = schedules(id)
        sch.SetData(MonthCalendar1.SelectionStart, txtMemo.Text)
        lbxDisp.Items(id) = sch.DispSchedule()

    End Sub

    Private Sub lbxDisp_SelectedIndexChanged(sender As Object,
                    e As EventArgs) Handles lbxDisp.SelectedIndexChanged

        Dim id As Integer = lbxDisp.SelectedIndex
        If id >= 0 Then

            Dim sch As Schedule = schedules(id)
            MonthCalendar1.SetSelectionRange(sch.GetSchDate, sch.GetSchDate)
            txtMemo.Text = sch.GetMemo()

        End If

    End Sub

End Class
```


【Schedule.vb】

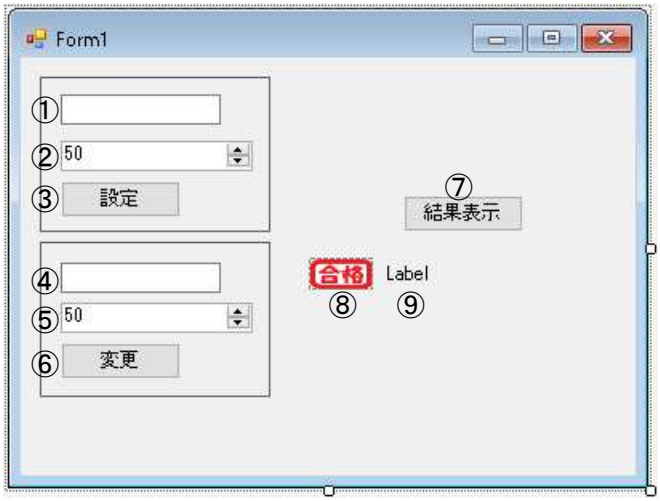
```
Public Class Schedule

    Dim schDate As Date
    Dim memo As String

    Sub New(tmpSchDate As Date, tmpMemo As String)

        schDate = tmpSchDate
        memo = tmpMemo

    End Sub

    Sub SetData(tmpSchDate As Date, tmpMemo As String)

        schDate = tmpSchDate
        memo = tmpMemo

    End Sub

    Function DispSchedule() As String

        Return "日付：" & schDate & "  予定：" & memo

    End Function

    Function GetSchDate() As Date

        Return schDate

    End Function

    Function GetMemo() As String

        Return memo

    End Function
End Class
```

課題（アクセス修飾子）

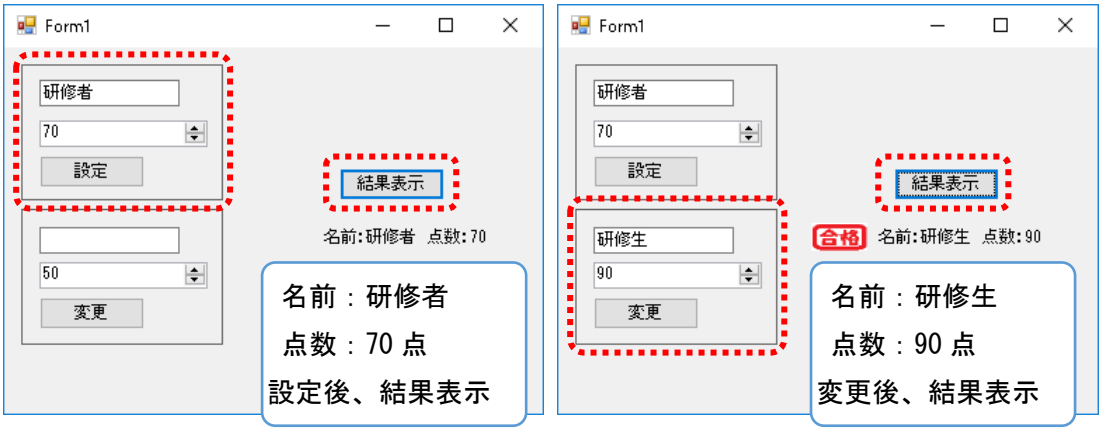
フォームもクラスです。ウィンドウのデザインを持っており、イベントと連動する特殊な機能を持っているクラスです。ソースコードの最初の行に「Class Form1」と記述されていることからも確認できます。よって、フォームから別のクラスのフィールドやメソッドを利用するときにもアクセス修飾子は影響します。

サンプル 3

【デザイン例】



【実行結果例】



番号	コントロール名	プロパティ名	プロパティ値
①	TextBox	(Name)	txtSetName
②	NumericUpDown	(Name)	nudSetScore
		Increment	5
		Maximum	100
		Minimum	0
		Value	50
③	Button	(Name)	btnSet
		Text	設定
④	TextBox	(Name)	txtUpdName
⑤	NumericUpDown	(Name)	nudUpdScore
		Increment	5
		Maximum	100
		Minimum	0
		Value	50
⑥	Button	(Name)	btnUpd
		Text	変更
⑦	Button	(Name)	btnResultDisp
		Text	結果表示
⑧	PictureBox	(Name)	picResult
		Image	（合格.png）
		SizeMode	StretchImage
		Visible	False
⑨	Label	(Name)	lblResult
		Visible	False

フィールド		
student	Student 型	Student のオブジェクトの参照を格納するための変数

コントロール	③	イベント	Click
Student クラスのオブジェクトを生成し、フィールド変数 student に格納する。 その際、コンストラクタを利用して①と②に入力された値を、そのオブジェクトへ設定する。			
コントロール	⑥	イベント	Click
④と⑤に入力された値を用いて、フィールド変数 student で参照しているオブジェクトの内容を更新する。			
コントロール	⑦	イベント	Click
フィールド変数 student で参照しているオブジェクトの内容を⑧と⑨に反映させる。 ⑧は student のフィールド変数 score が 80 以上である場合に表示させる。 ⑨は student の Disp()メソッドを利用して内容を表示させる。			

クラス名	Student	アクセス修飾子	Public

フィールド

変数名	型	アクセス修飾子	内容
name	String 型	Public	名前を管理するための変数
score	Integer 型	Public	点数を管理するための変数

コンストラクタ

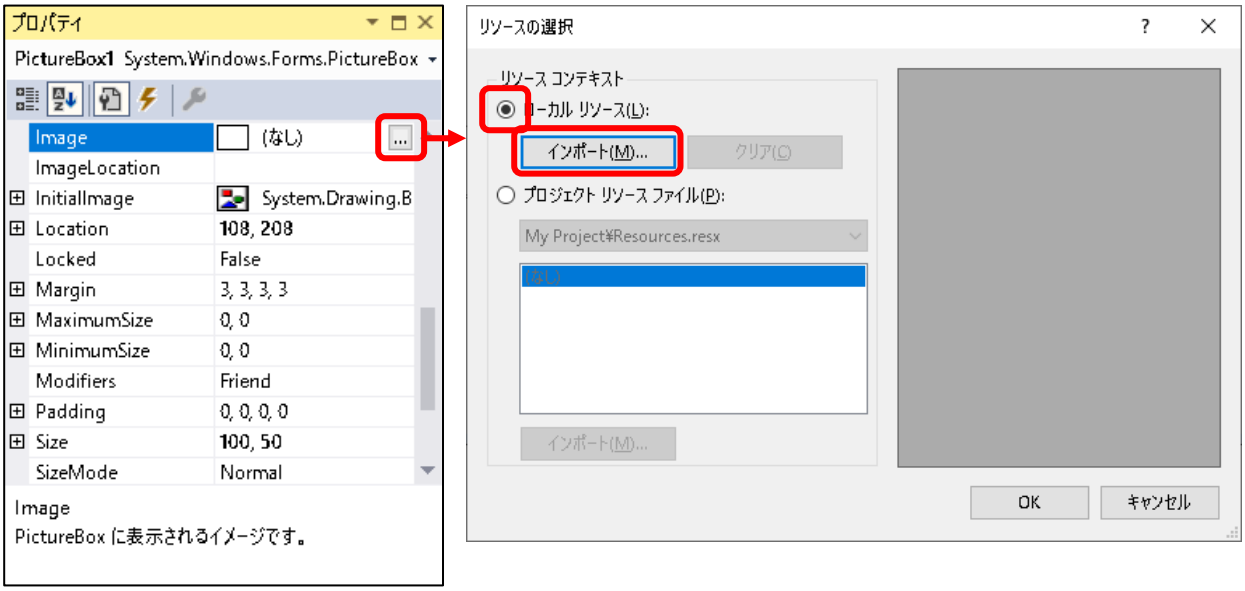
プロシージャ	Sub	アクセス修飾子	Public
引数	String 型, Integer 型		
第一引数で受け取った値をフィールドプロパティ name へ格納する。 第二引数で受け取った値をフィールドプロパティ score へ格納する。			

メソッド

プロシージャ	Sub	アクセス修飾子	Public
メソッド名	SetData	戻り値の型	（なし）
引数	String 型, Integer 型		
第一引数で受け取った値をフィールドプロパティ name へ格納する。 第二引数で受け取った値をフィールドプロパティ score へ格納する。			
プロシージャ	Function	アクセス修飾子	Public
メソッド名	Disp	戻り値の型	String 型
引数	（なし）		
フィールドプロパティ name および score をもとに表示文字列を返す。 表示例：「　名前：研修生　点数：60　」			

PictureBox

PictureBox は画像を表示するコントロールです。アイコンなどを表示する場合に利用します。Image プロパティから下図のようにファイルを選択します。



画像のサイズとコントロールのサイズは一致していなくても問題ありません。SizeMode プロパティに下記の設定をすることで調整可能です。

プロパティ値	挙動
Normal	何も調整しない
StretchImage	コントロールの大きさに画像が自動調整される
AutoSize	コントロールが画像の大きさに自動調整される
CenterImage	コントロールの中心に画像を表示し、画像とコントロールの大きさは調整しない
Zoom	画像サイズの縦横の比率を維持したままコントロールの大きさに画像を合わせる

実装 ExerciseSample03

```
Public Class Form1

    Dim student As Student

    Private Sub btnSet_Click(sender As Object,
                             e As EventArgs) Handles btnSet.Click

        student = New Student(txtSetName.Text, nudSetScore.Value)

    End Sub

    Private Sub btnUpdate_Click(sender As Object,
                                e As EventArgs) Handles btnUpdate.Click

        student.SetData(txtUpdName.Text, nudUpdScore.Value)

    End Sub

    Private Sub btnResultDisp_Click(sender As Object,
                            e As EventArgs) Handles btnResultDisp.Click

        If student.score >= 80 Then
            picResult.Visible = True
        Else
            picResult.Visible = False
        End If

        lblResult.Text = student.Disp()
        lblResult.Visible = True

    End Sub

End Class
```

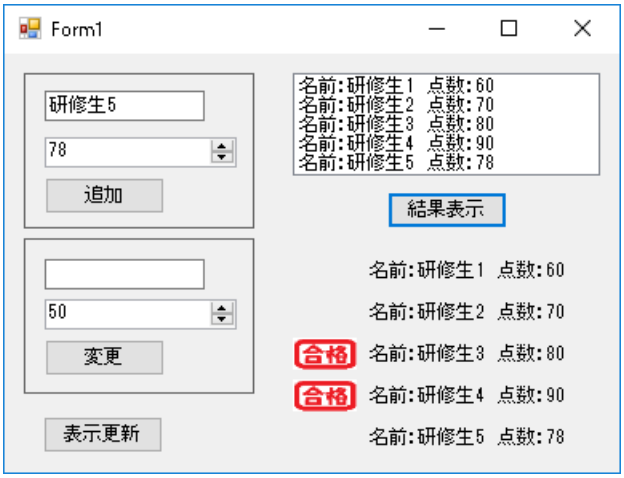
【Student.vb】

```
Public Class Student

    Public name As String
    Public score As Integer

    Public Sub SetData(tmpName As String, tmpScore As Integer)

        name = tmpName
        score = tmpScore

    End Sub

    Public Function Disp() As String

        Return "名前：" & name & "  点数：" & score

    End Function

    Public Sub New(tmpName As String, tmpScore As Integer)

        name = tmpName
        score = tmpScore

    End Sub

End Class
```


課題 3

【デザイン例】

【実行結果例】



番号	コントロール名	プロパティ名	プロパティ値
①	TextBox	(Name)	txtAddName
②	NumericUpDown	(Name)	nudAddScore
		Increment	5
		Value	50
③	Button	(Name)	btnAdd
		Text	追加
④	TextBox	(Name)	txtUpdName
⑤	NumericUpDown	(Name)	nudUpdScore
		Increment	5
		Value	50
⑥	Button	(Name)	btnUpdate
		Text	変更
⑦	Button	(Name)	btnDisp
		Text	表示更新
⑧	ListBox	(Name)	lbxDisp
⑨	Button	(Name)	btnResultDisp
		Text	結果表示
⑩～⑭	PictureBox	(Name)	picResult1 , picResult2 , picResult3 , picResult4 , picResult5
		SizeMode	StretchImage
		Visible	False
		Image	（合格.png）
⑮～⑲	Label	(Name)	lblResult1 , lblResult2 , lblResult3 , lblResult4 , lblResult5
		Text	（任意）
		Visible	False

フィールド		
students	Student 型の配列	Student のオブジェクトの参照を格納するための配列。要素数は 5 とする（要素番号は 0～4）
idx	Integer 型	上記の配列の要素番号を管理するための変数

コントロール	③	イベント	Click
Student クラスのオブジェクトを生成し、フィールド変数 students に追加する。 その際、コンストラクタを利用して①と②に入力された値を、そのオブジェクトへ設定する。 （このボタンが押されるたびにオブジェクトを生成して配列へ格納する。5 つのオブジェクトを格納したあとはメッセージを表示してオブジェクトを生成させない。）			
コントロール	⑥	イベント	Click
④と⑤に入力された値を用いて、フィールド変数 students で参照しているオブジェクトの内容を更新する。更新対象のオブジェクトは⑧で選択されているものとする。 選択されていない場合はメッセージを表示して内容の更新はしない。			
コントロール	⑦	イベント	Click
フィールド変数 students の Disp()メソッドを利用して⑧に内容を表示させる。 1 件を 1 行で表示させる。			
コントロール	⑨	イベント	Click
フィールド変数 students の内容を⑮～⑲へ表示させる。その際、score が 80 以上の場合は⑩～⑭のうち対応するものを表示させる			

クラス名	Student	アクセス修飾子	Public

フィールド

変数名	型	アクセス修飾子	内容
name	String 型	Public	名前を管理するための変数
score	Integer 型	Public	点数を管理するための変数

コンストラクタ

プロシージャ	Sub	アクセス修飾子	Public
引数	String 型, Integer 型		
第一引数で受け取った値をフィールドプロパティ name へ格納する。 第二引数で受け取った値をフィールドプロパティ score へ格納する。			

メソッド

プロシージャ	Sub	アクセス修飾子	Public
メソッド名	SetData	戻り値の型	（なし）
引数	String 型, Integer 型		
第一引数で受け取った値をフィールドプロパティ name へ格納する。 第二引数で受け取った値をフィールドプロパティ score へ格納する。			
プロシージャ	Function	アクセス修飾子	Public
メソッド名	Disp	戻り値の型	String 型
引数	（なし）		
フィールドプロパティ name および score をもとに表示文字列を返す。 表示例：「　名前：研修生　点数：60　」			
フィールドプロパティ name を返す。			

コントロールの配列

コントロールもプログラム上ではデータです。配置されたコントロールはオブジェクトとして扱えます。オブジェクトを参照するための名前（変数名）は Name プロパティと同じです。変数として扱えるということは配列としても扱えるということです。なぜなら、配列は変数をいくつか集めたものだからです。

前述のサンプルでは、5 つの PictureBox のオブジェクトをまとめて配列としています。5 つの Label についても同様です。配列のインデックス番号を変化させることで、配列内のそれぞれのオブジェクトを扱うことができます。

解答例 **Exercise03**

【Form1.vb】

```
Public Class Form1

    Dim students(4) As Student
    Dim idx As Integer = 0

    Private Sub btnAdd_Click(sender As Object,
                             e As EventArgs) Handles btnAdd.Click

        If idx <= 4 Then

            Dim std As Student = New Student(txtAddName.Text, nudAddScore.Value)

            students(idx) = std
            idx += 1
        Else
            MsgBox("すでに 5 名登録済みです")
        End If

    End Sub

    Private Sub btnUpdate_Click(sender As Object,
                                e As EventArgs) Handles btnUpdate.Click
        If lbxDisp.SelectedIndex < 0 Then
            MsgBox("変更対象を選択してください")
            Exit Sub
        End If

        Dim std As Student = students(lbxDisp.SelectedIndex)
        std.SetData(txtUpdName.Text, nudUpdScore.Value)

    End Sub

    Private Sub btnDisp_Click(sender As Object,
                              e As EventArgs) Handles btnDisp.Click

        lbxDisp.Items.Clear()
```

```
        For cnt As Integer = 0 To idx - 1
            lbxDisp.Items.Add(students(cnt).Disp())
        Next

    End Sub

    Private Sub btnResultDisp_Click(sender As Object,
                            e As EventArgs) Handles btnResultDisp.Click
        Dim arrayPic() As PictureBox =
            {picResult1, picResult2, picResult3, picResult4, picResult5}

        Dim arrayLbl() As Label =
            {lblResult1, lblResult2, lblResult3, lblResult4, lblResult5}

        For cnt As Integer = 0 To idx - 1

            Dim std As Student = students(cnt)

            If std.score >= 80 Then
                arrayPic(cnt).Visible = True
            Else
                arrayPic(cnt).Visible = False
            End If

            arrayLbl(cnt).Text = std.Disp()
            arrayLbl(cnt).Visible = True

        Next

    End Sub

End Class
```


【Student.vb】

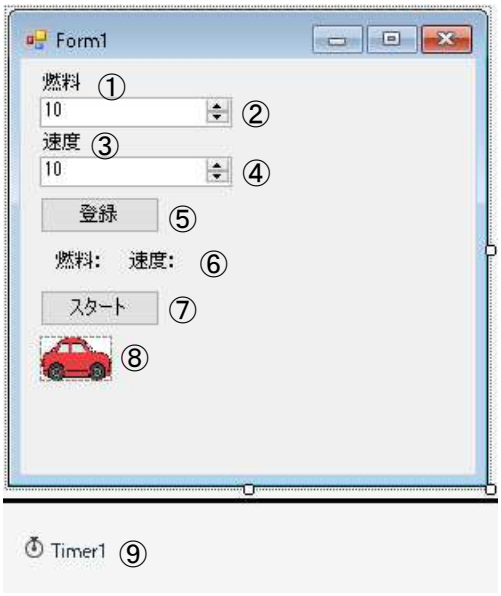
```
Public Class Student

    Public name As String
    Public score As Integer

    Public Sub SetData(tmpName As String, tmpScore As Integer)

        name = tmpName
        score = tmpScore

    End Sub

    Public Function Disp() As String

        Return "名前：" & name & "  点数：" & score

    End Function

    Public Sub New(tmpName As String, tmpScore As Integer)

        name = tmpName
        score = tmpScore

    End Sub

End Class
```

課題（プロパティ）

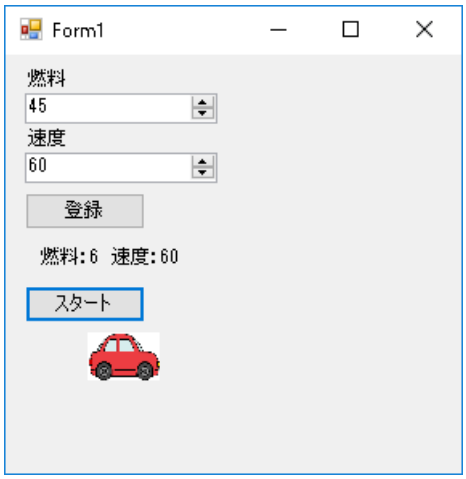
本課題では Car クラスを用いて車をフォーム上で走らせるプログラムを作っていきます。車が走っているように見せるには、車の画像を移動させますが、この時に必要になるのがタイマーです。タイマーを利用して、少しずつ右へ移動させると、パラパラ漫画のように車が移動しているように見えます。タイマーの利用については後述しています。

サンプル 4

【デザイン例】



【実行結果例】



番号	コントロール名	プロパティ名	プロパティ値
①	Label	Text	燃料
②	NumericUpDown	(Name)	nudFuel
		Increment	5
		Value	10
③	Label	Text	速度
④	NumericUpDown	(Name)	nudSpeed
		Increment	5
		Value	10
⑤	Button	(Name)	btnAdd
		Text	登録

⑥	Label	(Name)	lblDisp
		Text	（任意）
⑦	Button	(Name)	btnStart
		Text	スタート
⑧	PictureBox	(Name)	pctCar
		Image	（車.png）
		SizeMode	StretchImage
⑨	Timer	ー	ー

フィールド		
car	Car 型	Car クラスのオブジェクトの参照を格納するための変数

コントロール	⑤	イベント	Click
Car クラスのオブジェクトを生成し、フィールド変数 car へ格納する。(Car クラスについては後述) ②と④に入力された値を、SetData()メソッドでそのオブジェクトへ設定する。 フィールド変数 car の Disp()メソッドを用いて現在の情報を⑥へ表示させる。			
コントロール	⑦	イベント	Click
⑧の表示位置を初期値に戻す。 ⑨の Interval を次の式を用いて設定する。「 210 - 速度 * 2 」 ⑨のタイマーを開始する。			
コントロール	⑨	イベント	Tick
フィールド変数 car の燃料を 1 減らす。 ⑧の表示位置を右に 1 ずらす。 フィールド変数 car の Disp()メソッドの結果を⑥へ表示させる。 もし燃料が 0 になった場合は⑨のタイマーを停止させる。			

クラス名	Car	アクセス修飾子	Public

フィールド

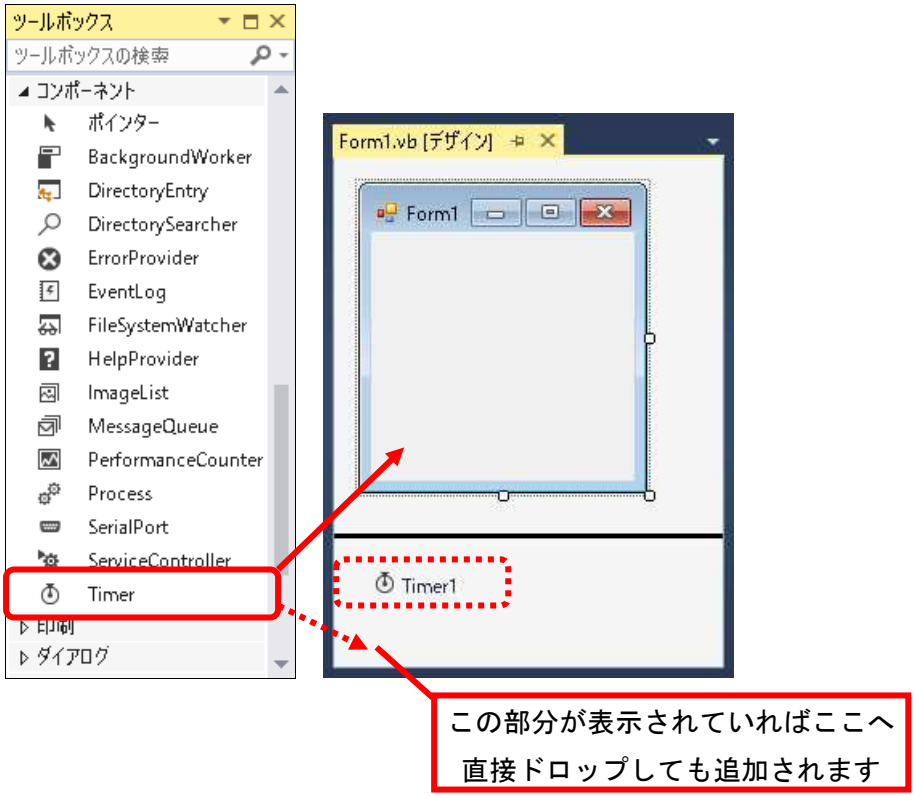
プロパティ名	型	内容
Fuel	Integer 型	燃料の情報を管理するためのプロパティ
Speed	Integer 型	速度の情報を管理するためのプロパティ

メソッド

プロシージャ	Sub	アクセス修飾子	Public
メソッド名	SetData	戻り値の型	（なし）
引数	Integer 型 , Integer 型		
第一引数で受け取った値をフィールドプロパティ Fuel へ格納する。 第二引数で受け取った値をフィールドプロパティ Speed へ格納する。			
プロシージャ	Function	アクセス修飾子	Public
メソッド名	Disp	戻り値の型	String 型
引数	（なし）		
フィールドプロパティ Fuel および Speed をもとに表示文字列を返す。 表示例：「　燃料：100　速度：100　」			

Timer

Timer は一定の時間が経ったら Tick イベントを発生させるコントロールです。Button などと同じようにフォームへドロップして追加しますが、フォーム上へは何も表示されません。フォームの下のエリアにアイコンと名前が表示されますので、プロパティの設定はこの部分をクリックします。表示物を持たないので、大きさや色といったプロパティはありません。Interval プロパティで Tick イベントを発生させるタイミングをミリ秒単位で設定します。1000 と設定した場合、一秒ごとに Tick イベントが発生します。また、Start()メソッドでタイマーを開始し、Stop()メソッドで停止します。Stop()メソッドは一時停止ではなく、キャンセルのような動作で、再度 Start()した場合は途中からではなく最初から数え直します。



実装 ExerciseSample04

【Form1.vb】

```
Public Class Form1

    Dim car As Car = New Car()

    Private Sub btnAdd_Click(sender As Object,
                             e As EventArgs) Handles btnAdd.Click

        car.SetData(nudFuel.Value, nudSpeed.Value)
        lblDisp.Text = car.Disp()

    End Sub

    Private Sub btnStart_Click(sender As Object,
                               e As EventArgs) Handles btnStart.Click

        pctCar.Left = 12
        Timer1.Interval = 210 - car.Speed * 2
        Timer1.Start()

    End Sub

    Private Sub Timer1_Tick(sender As Object,
                            e As EventArgs) Handles Timer1.Tick

        car.Fuel -= 1
        pctCar.Left += 1
        lblDisp.Text = car.Disp()

        If car.Fuel <= 0 Then
            Timer1.Stop()
        End If

    End Sub

End Class
```

フォーム上へ配置した位置(Location.X プロパティ)を設定してください

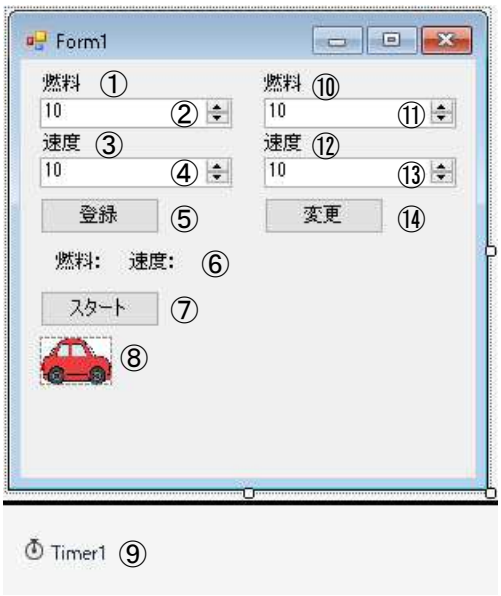
【Car.vb】

```
Public Class Car

    Public Property Fuel As Integer
    Public Property Speed As Integer

    Public Sub SetData(tmpFuel As Integer, tmpSpeed As Integer)

        Fuel = tmpFuel
        Speed = tmpSpeed

    End Sub

    Public Function Disp() As String

        Return "  燃料:" & Fuel & "  速度:" & Speed

    End Function

End Class
```

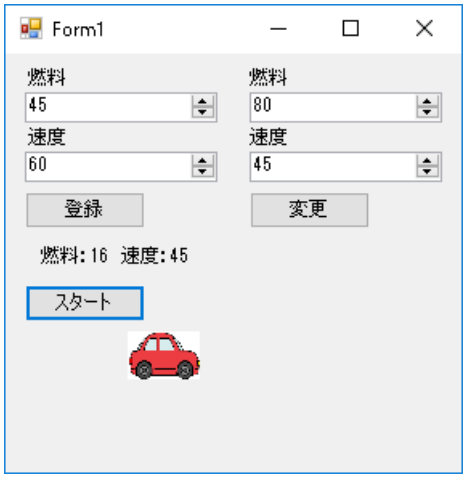
課題 4

先のサンプルに燃料と速度の変更機能を追加します。

【デザイン例】



【実行結果例】



番号	コントロール名	プロパティ名	プロパティ値
①	Label	Text	燃料
②	NumericUpDown	(Name)	nudFuel
		Increment	5
		Value	10
③	Label	Text	速度
④	NumericUpDown	(Name)	nudSpeed
		Increment	5
		Value	10
⑤	Button	(Name)	btnAdd
		Text	登録
⑥	Label	(Name)	lblDisp
		Text	（任意）
⑦	Button	(Name)	btnStart
		Text	スタート
⑧	PictureBox	(Name)	pctCar
		Image	（車.png）
		SizeMode	StretchImage
⑨	Timer	―	―

⑩	Label	Text	燃料
⑪	NumericUpDown	(Name)	nudUpdFuel
		Increment	5
		Value	10
⑫	Label	Text	速度
⑬	NumericUpDown	(Name)	nudUpdSpeed
		Increment	5
		Value	10
⑭	Button	(Name)	btnUpdate
		Text	変更

フィールド		
car	Car 型	Car のオブジェクトの参照を格納するための変数

コントロール	⑤	イベント	Click
Car クラスのオブジェクトを生成し、フィールド変数 car へ格納する。(Car クラスについては後述) ②と④に入力された値を、SetData()メソッドでそのオブジェクトへ設定する。 そのオブジェクトの Disp()メソッドの結果を⑥へ表示させる。			
コントロール	⑦	イベント	Click
⑧の表示位置を初期値に戻す。 ⑨の Interval を次の式を用いて設定する。「 210 - 速度 * 2 」 ⑨のタイマーを開始する。			
コントロール	⑨	イベント	Tick
フィールド変数 car の燃料を 1 減らす。 ⑧の表示位置を右に 1 ずらす。 フィールド変数 car の Disp()メソッドを用いて現在の情報を⑥へ表示させる。 もし燃料が 0 になった場合は⑨のタイマーを停止させる。			
コントロール	⑭	イベント	Click
⑪と⑬に入力された値を、SetData()メソッドでフィールド変数 car が参照しているオブジェクトへ設定する。 ⑥にフィールド変数 car の Disp()メソッドを用いて現在の情報を表示させる。			

クラス名	Car	アクセス修飾子	Public

フィールド

プロパティ名	型	内容
Fuel	Integer 型	燃料の情報を管理するためのプロパティ
Speed	Integer 型	速度の情報を管理するためのプロパティ

メソッド

プロシージャ	Sub	アクセス修飾子	Public
メソッド名	SetData	戻り値の型	（なし）
引数	Integer 型 , Integer 型		
第一引数で受け取った値をフィールドプロパティ Fuel へ格納する。 第二引数で受け取った値をフィールドプロパティ Speed へ格納する。			
プロシージャ	Function	アクセス修飾子	Public
メソッド名	Disp	戻り値の型	String 型
引数	（なし）		
フィールドプロパティ Fuel および Speed をもとに表示文字列を返す。 表示例：「　燃料：100　速度：100　」			

解答例 Exercise04

【Form1.vb】

```
Public Class Form1

    Dim car As Car

    Private Sub btnAdd_Click(sender As Object,
                             e As EventArgs) Handles btnAdd.Click

        car = New Car()
        car.SetData(nudFuel.Value, nudSpeed.Value)

        lblDisp.Text = car.Disp()

    End Sub

    Private Sub btnStart_Click(sender As Object,
                               e As EventArgs) Handles btnStart.Click

        pctCar.Left = 12
        Timer1.Interval = 210 - car.Speed * 2
        Timer1.Start()

    End Sub

    Private Sub Timer1_Tick(sender As Object,
                            e As EventArgs) Handles Timer1.Tick

        car.Fuel -= 1
        pctCar.Left += 1
        lblDisp.Text = car.Disp()

        If car.Fuel <= 0 Then
            Timer1.Stop()
        End If

    End Sub

    Private Sub btnUpdate_Click(sender As Object,
                                e As EventArgs) Handles btnUpdate.Click

        car.SetData(nudUpdFuel.Value, nudUpdSpeed.Value)

        lblDisp.Text = car.Disp()

    End Sub

End Class
```

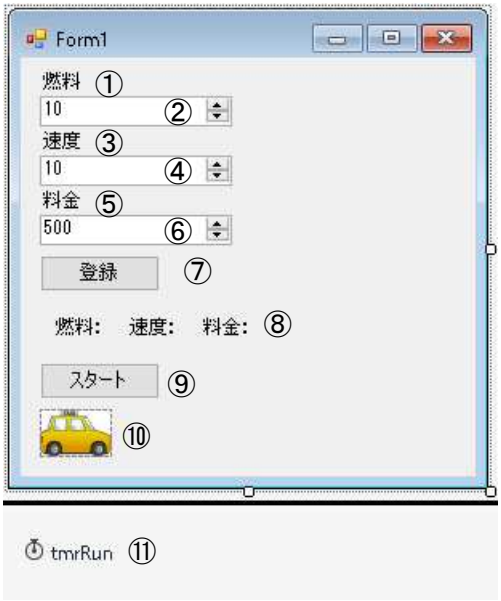
【Car.vb】

```
Public Class Car

    Public Property Fuel As Integer
    Public Property Speed As Integer

    Public Sub SetData(tmpFuel As Integer, tmpSpeed As Integer)

        Fuel = tmpFuel
        Speed = tmpSpeed

    End Sub

    Public Function Disp() As String

        Return "  燃料：" & Fuel & "  速度：" & Speed

    End Function

End Class
```


課題（継承）

サンプル 5

【デザイン例】



【実行結果例】

番号	コントロール名	プロパティ名	プロパティ値
①	Label	Text	燃料
②	NumericUpDown	(Name)	nudFuel
		Increment	5
		Value	10
③	Label	Text	速度
④	NumericUpDown	(Name)	nudSpeed
		Increment	5
		Value	10
⑤	Label	Text	料金
⑥	NumericUpDown	(Name)	nudFare
		Increment	100
		Maximum	10000
		Minimum	500
		Value	500
⑦	Button	(Name)	btnAdd
		Text	登録

⑧	Label	(Name)	lblDisp
		Text	（任意）
⑨	Button	(Name)	btnStart
		Text	スタート
⑩	PictureBox	(Name)	pctCar
		Image	（タクシー.png）
		SizeMode	StretchImage
⑪	Timer	(Name)	tmrRun

フィールド		
taxi	Taxi 型	Taxi クラスのオブジェクトの参照を格納するための変数

コントロール	⑦	イベント	Click
Taxi クラスのオブジェクトを生成し、フィールド変数 taxi へ格納する。(Taxi クラスについては後述) ②と④と⑥に入力された値を、SetTaxiData()メソッドでそのオブジェクトへ設定する。 フィールド変数 taxi の Disp()メソッドと DispFare()メソッドを用いて現在の情報を⑧へ表示させる。			
コントロール	⑨	イベント	Click
⑩の表示位置を初期値に戻す。 ⑪の Interval を次の式を用いて設定する。「 210 － 速度 ＊ 2 」 ⑪のタイマーを開始する。			
コントロール	⑪	イベント	Tick
フィールド変数 taxi の燃料を 1 減らす。 フィールド変数 taxi の IncreaseFare()メソッドを用いて料金を増やす。 ⑩の表示位置を右に 1 ずらす。 フィールド変数 taxi の Disp()メソッドと DispFare()メソッドを用いて現在の情報を⑧へ表示させる。 もし燃料が 0 になった場合は⑪のタイマーを停止させる。			

クラス名	Car	アクセス修飾子	Public

フィールド

プロパティ名	型	内容
Fuel	Integer 型	燃料の情報を管理するためのプロパティ
Speed	Integer 型	速度の情報を管理するためのプロパティ

メソッド

プロシージャ	Sub	アクセス修飾子	Public
メソッド名	SetData	戻り値の型	（なし）
引数	Integer 型 , Integer 型		
第一引数で受け取った値をフィールドプロパティ Fuel へ格納する。 第二引数で受け取った値をフィールドプロパティ Speed へ格納する。			
プロシージャ	Function	アクセス修飾子	Public
メソッド名	Disp	戻り値の型	String 型
引数	（なし）		
フィールドプロパティ Fuel および Speed をもとに表示文字列を返す。 表示例：「　燃料：100　速度：100　」			

クラス名	Taxi	アクセス修飾子	Public
継承	Car		

フィールド

プロパティ名	型	内容
Fare	Integer 型	料金の情報を管理するためのプロパティ

メソッド

プロシージャ	Sub	アクセス修飾子	Public
メソッド名	SetTaxiData	戻り値の型	（なし）
引数	Integer 型 , Integer 型 , Integer 型		
第一引数で受け取った値をフィールドプロパティ Fuel へ格納する。 第二引数で受け取った値をフィールドプロパティ Speed へ格納する。 第三引数で受け取った値をフィールドプロパティ Fare へ格納する。			
プロシージャ	Function	アクセス修飾子	Public
メソッド名	DispFare	戻り値の型	String 型
引数	（なし）		
フィールドプロパティ Fare をもとに表示文字列を返す。 表示例：「　料金：100　」			
プロシージャ	Sub	アクセス修飾子	Public
メソッド名	IncreaseFare	戻り値の型	（なし）
引数	（なし）		
フィールドプロパティ Fare に 100 加算する。			

実装 ExerciseSample05

【Form1.vb】

```
Public Class Form1

    Dim taxi As Taxi

    Private Sub btnAdd_Click(sender As Object,
                             e As EventArgs) Handles btnAdd.Click

        taxi = New Taxi()
        taxi.SetTaxiData(nudFuel.Value, nudSpeed.Value, nudFare.Value)

        'taxi.SetData(nudFuel.Value, nudSpeed.Value)
        'taxi.Fare = nudFare.Value

        lblDisp.Text = taxi.Disp() & taxi.DispFare()

    End Sub

    Private Sub btnStart_Click(sender As Object,
                               e As EventArgs) Handles btnStart.Click

        pctCar.Left = 12
        tmrRun.Interval = 210 - taxi.Speed * 2
        tmrRun.Start()

    End Sub

    Private Sub tmrRun_Tick(sender As Object,
                            e As EventArgs) Handles tmrRun.Tick

        taxi.Fuel -= 1
        taxi.IncreaseFare()
        pctCar.Left += 1
        lblDisp.Text = taxi.Disp() & taxi.DispFare()

        If taxi.Fuel <= 0 Then
            tmrRun.Stop()
        End If

    End Sub

End Class
```

【Car.vb】

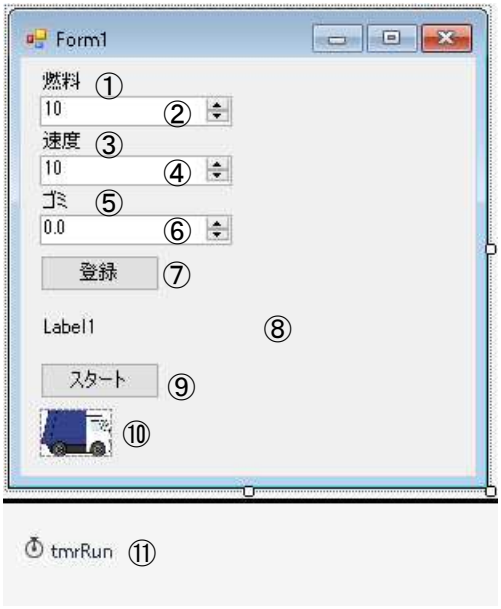
```
Public Class Car

    Public Property Fuel As Integer
    Public Property Speed As Integer

    Public Sub SetData(tmpFuel As Integer, tmpSpeed As Integer)

        Fuel = tmpFuel
        Speed = tmpSpeed

    End Sub

    Public Function Disp() As String

        Return "  燃料：" & Fuel & "  速度：" & Speed

    End Function

End Class
```

【Taxi.vb】

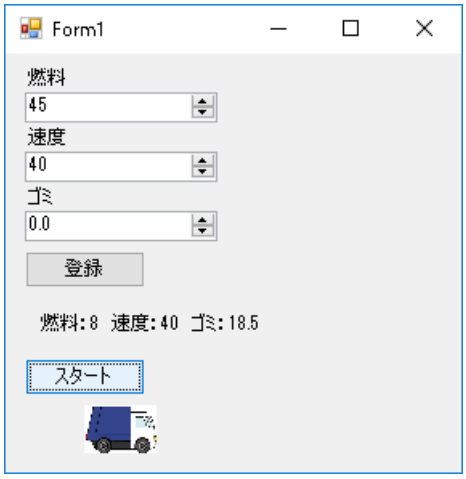
```
Public Class Taxi
    Inherits Car

    Public Property Fare As Integer

    Public Sub SetTaxiData(tmpFuel As Integer,
                           tmpSpeed As Integer, tmpFare As Integer)

        Fuel = tmpFuel
        Speed = tmpSpeed
        Fare = tmpFare

    End Sub

    Public Function DispFare() As String

        Return "  料金：" & Fare

    End Function

    Public Sub IncreaseFare()
        Fare += 100
    End Sub

End Class
```


課題 5

【デザイン例】



【実行結果例】



番号	コントロール名	プロパティ名	プロパティ値
①	Label	Text	燃料
②	NumericUpDown	(Name)	nudFuel
		Increment	5
		Value	10
③	Label	Text	速度
④	NumericUpDown	(Name)	nudSpeed
		Increment	5
		Value	10
⑤	Label	Text	ゴミ
⑥	NumericUpDown	(Name)	nudGarbage
		DecimalPlaces	1
		Increment	0.5
⑦	Button	(Name)	btnAdd
		Text	登録
⑧	Label	(Name)	lblDisp
		Text	（任意）
⑨	Button	(Name)	btnStart
		Text	スタート

⑩	PictureBox	(Name)	pctCar
		Image	（ゴミ収集車.png）
		SizeMode	StretchImage
⑪	Timer	(Name)	tmrRun

フィールド		
gbg	GarbageTruck 型	GarbageTruck クラスのオブジェクトの参照を格納するための変数

コントロール	⑦	イベント	Click
GarbageTruck クラスのオブジェクトを生成し、フィールド変数 gbg へ格納する。(GarbageTruck クラスについては後述) ②と④と⑥に入力された値を、SetGarbageData()メソッドでそのオブジェクトへ設定する。 フィールド変数 gbg の Disp()メソッドと DispGarbage()メソッドを用いて現在の情報を⑧へ表示させる。			
コントロール	⑨	イベント	Click
⑩の表示位置を初期値に戻す。 ⑪の Interval を次の式を用いて設定する。「　210 － 速度 ＊ 2　」 ⑪のタイマーを開始する。			
コントロール	⑪	イベント	Tick
フィールド変数 gbg の燃料を 1 減らす。 フィールド変数 gbg の CollectGarbage()メソッドを用いてゴミを増やす。 ⑩の表示位置を右に 1 ずらす。 フィールド変数 gbg の Disp()メソッドと DispGarbage()メソッドを用いて現在の情報を⑧へ表示させる。 もし燃料が 0 になった場合は⑪のタイマーを停止させる。			

クラス名	Car	アクセス修飾子	Public

フィールド

プロパティ名	型	内容
Fuel	Integer 型	燃料の情報を管理するためのプロパティ
Speed	Integer 型	速度の情報を管理するためのプロパティ

メソッド

プロシージャ	Sub	アクセス修飾子	Public
メソッド名	SetData	戻り値の型	（なし）
引数	Integer 型 , Integer 型		
第一引数で受け取った値をフィールドプロパティ Fuel へ格納する。 第二引数で受け取った値をフィールドプロパティ Speed へ格納する。			
プロシージャ	Function	アクセス修飾子	Public
メソッド名	Disp	戻り値の型	String 型
引数	（なし）		
フィールドプロパティ Fuel および Speed をもとに表示文字列を返す。 表示例：「　燃料：100　速度：100　」			

クラス名	GarbageTruck	アクセス修飾子	Public
継承	Car		

フィールド

プロパティ名	型	内容
Garbage	Double 型	ゴミの情報を管理するためのプロパティ

メソッド

プロシージャ	Sub	アクセス修飾子	Public
メソッド名	SetGarbageData	戻り値の型	（なし）
引数	Integer 型 , Integer 型 , Double 型		
第一引数で受け取った値をフィールドプロパティ Fuel へ格納する。 第二引数で受け取った値をフィールドプロパティ Speed へ格納する。 第三引数で受け取った値をフィールドプロパティ Garbage へ格納する。			
プロシージャ	Function	アクセス修飾子	Public
メソッド名	DispGarbage	戻り値の型	String 型
引数	（なし）		
フィールドプロパティ Garbage をもとに表示文字列を返す。 表示例：「　ゴミ：100　」			
プロシージャ	Sub	アクセス修飾子	Public
メソッド名	CollectGarbage	戻り値の型	（なし）
引数	（なし）		
フィールドプロパティ Garbage に 0.5 加算する。			

解答例 Exercise05

【Form1.vb】

```
Public Class Form1

    Dim gbg As GarbageTruck

    Private Sub btnAdd_Click(sender As Object,
                             e As EventArgs) Handles btnAdd.Click

        gbg = New GarbageTruck()
        gbg.SetGarbageData(nudFuel.Value, nudSpeed.Value, nudGarbage.Value)

        lblDisp.Text = gbg.Disp() & gbg.DispGarbage()

    End Sub

    Private Sub btnStart_Click(sender As Object,
                               e As EventArgs) Handles btnStart.Click

        pctCar.Left = 12
        tmrRun.Interval = 210 - gbg.Speed * 2
        tmrRun.Start()

    End Sub

    Private Sub tmrRun_Tick(sender As Object,
                            e As EventArgs) Handles tmrRun.Tick

        gbg.Fuel -= 1
        gbg.CollectGarbage()
        pctCar.Left += 1
        lblDisp.Text = gbg.Disp() & gbg.DispGarbage()

        If gbg.Fuel <= 0 Then
            tmrRun.Stop()
        End If

    End Sub

End Class
```

【Car.vb】

```
Public Class Car

    Public Property Fuel As Integer
    Public Property Speed As Integer

    Public Sub SetData(tmpFuel As Integer, tmpSpeed As Integer)

        Fuel = tmpFuel
        Speed = tmpSpeed

    End Sub

    Public Function Disp() As String

        Return "  燃料：" & Fuel & "  速度：" & Speed

    End Function

End Class
```

【GarbageTruck.vb】

```
Public Class GarbageTruck
    Inherits Car

    Public Property Garbage As Double

    Public Sub SetGarbageData(tmpFuel As Integer,
                              tmpSpeed As Integer, tmpGarbage As Double)

        Fuel = tmpFuel
        Speed = tmpSpeed
        Garbage = tmpGarbage

    End Sub

    Public Function DispGarbage() As String

        Return "  ゴミ：" & Garbage

    End Function

    Public Sub CollectGarbage()

        Garbage += 0.5

    End Sub

End Class
```


課題（オーバーライド）

サンプル 6

【デザイン例】

【実行結果例】

番号	コントロール名	プロパティ名	プロパティ値
①	Label	Text	燃料
②	NumericUpDown	(Name)	nudFuel
		Increment	5
		Value	10
③	Label	Text	速度
④	NumericUpDown	(Name)	nudSpeed
		Increment	5
		Value	10
⑤	Label	Text	料金
⑥	NumericUpDown	(Name)	nudFare
		Increment	100
		Maximum	10000
		Minimum	500
		Value	500

⑦	Button	(Name)	btnAdd
		Text	登録
⑧	Label	(Name)	lblDisp
		Text	（任意）
⑨	Button	(Name)	btnStart
		Text	スタート
⑩	PictureBox	(Name)	pctCar
		Image	（タクシー.png）
		SizeMode	StretchImage
⑪	Timer	(Name)	tmrRun

フィールド		
taxi	Taxi 型	Taxi クラスのオブジェクトの参照を格納するための変数

コントロール	⑦	イベント	Click
Taxi クラスのオブジェクトを生成し、フィールド変数 taxi へ格納する。(Taxi クラスについては後述) ②と④に⑥と入力された値を、SetTaxiData()メソッドでそのオブジェクトへ設定する。 フィールド変数 taxi の Disp()メソッドを用いて現在の情報を⑧へ表示させる。			
コントロール	⑨	イベント	Click
⑩の表示位置を初期値に戻す。 ⑪の Interval を次の式を用いて設定する。「 210 - 速度 * 2 」 ⑪のタイマーを開始する。			
コントロール	⑪	イベント	Tick
フィールド変数 taxi の燃料を 1 減らす。 フィールド変数 taxi の IncreaseFare()メソッドを用いて料金を増やす。 ⑩の表示位置を右に 1 ずらす。 フィールド変数 taxi の Disp()メソッドを用いて現在の情報を⑧へ表示させる。 もし燃料が 0 になった場合は⑪のタイマーを停止させる。			

クラス名	Car	アクセス修飾子	Public

フィールド		

プロパティ名	型	内容
Fuel	Integer 型	燃料の情報を管理するためのプロパティ
Speed	Integer 型	速度の情報を管理するためのプロパティ

メソッド

プロシージャ	Sub	アクセス修飾子	Public
メソッド名	SetData	戻り値の型	（なし）
引数	Integer 型 , Integer 型		
第一引数で受け取った値をフィールドプロパティ Fuel へ格納する。 第二引数で受け取った値をフィールドプロパティ Speed へ格納する。			
プロシージャ	Function	アクセス修飾子	Public
メソッド名	Disp	戻り値の型	String 型
オーバーライド	Overridable		
引数	（なし）		
フィールドプロパティ Fuel および Speed をもとに表示文字列を返す。 表示例：「　燃料：100　速度：100　」			

クラス名	Taxi	アクセス修飾子	Public
継承	Car		

フィールド

プロパティ名	型	内容
Fare	Integer 型	料金の情報を管理するためのプロパティ

メソッド

プロシージャ	Sub	アクセス修飾子	Public
メソッド名	SetTaxiData	戻り値の型	（なし）
引数	Integer 型 , Integer 型 , Integer 型		
第一引数で受け取った値をフィールドプロパティ Fuel へ格納する。 第二引数で受け取った値をフィールドプロパティ Speed へ格納する。 第三引数で受け取った値をフィールドプロパティ Fare へ格納する。			
プロシージャ	Function	アクセス修飾子	Public
メソッド名	Disp	戻り値の型	String 型
オーバーライド	Overrides		
引数	（なし）		
基本クラスの Disp()メソッドの結果とフィールドプロパティ Fare をもとに表示文字列を返す。 表示例：「　燃料：100　速度：100　料金：100　」			
プロシージャ	Sub	アクセス修飾子	Public
メソッド名	IncreaseFare	戻り値の型	（なし）
引数	（なし）		
フィールドプロパティ Fare に 100 加算する。			

実装 ExerciseSample06

【Form1.vb】

```
Public Class Form1

    Dim taxi As Taxi

    Private Sub btnAdd_Click(sender As Object,
                             e As EventArgs) Handles btnAdd.Click

        taxi = New Taxi()
        taxi.SetTaxiData(nudFuel.Value, nudSpeed.Value, nudFare.Value)

        'taxi.SetData(nudFuel.Value, nudSpeed.Value)
        'taxi.Fare = nudFare.Value

        lblDisp.Text = taxi.Disp()

    End Sub

    Private Sub btnStart_Click(sender As Object,
                               e As EventArgs) Handles btnStart.Click

        pctCar.Left = 12
        tmrRun.Interval = 210 - taxi.Speed * 2
        tmrRun.Start()

    End Sub

    Private Sub tmrRun_Tick(sender As Object,
                            e As EventArgs) Handles tmrRun.Tick

        taxi.Fuel -= 1
        taxi.IncreaseFare()
        pctCar.Left += 1
        lblDisp.Text = taxi.Disp()

        If taxi.Fuel <= 0 Then
            tmrRun.Stop()
        End If

    End Sub

End Class
```

【Car.vb】

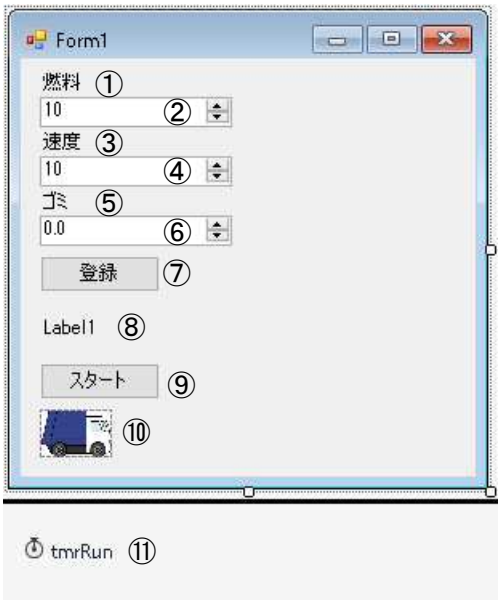
```
Public Class Car

    Public Property Fuel As Integer
    Public Property Speed As Integer

    Public Sub SetData(tmpFuel As Integer, tmpSpeed As Integer)

        Fuel = tmpFuel
        Speed = tmpSpeed

    End Sub

    Public Overridable Function Disp() As String

        Return "  燃料：" & Fuel & "  速度：" & Speed

    End Function

End Class
```

【Taxi.vb】

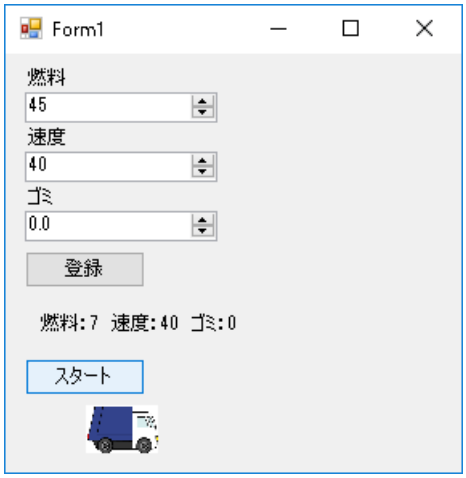
```
Public Class Taxi
    Inherits Car

    Public Property Fare As Integer

    Public Sub SetTaxiData(tmpFuel As Integer,
                           tmpSpeed As Integer, tmpFare As Integer)

        MyBase.SetData(tmpFuel, tmpSpeed)
        Fare = tmpFare

    End Sub

    Public Overrides Function Disp() As String

        Return MyBase.Disp() & "  料金：" & Fare

    End Function

    Public Sub IncreaseFare()

        Fare += 100

    End Sub

End Class
```


課題 6

【デザイン例】



【実行結果例】



番号	コントロール名	プロパティ名	プロパティ値
①	Label	Text	燃料
②	NumericUpDown	(Name)	nudFuel
		Increment	5
		Value	10
③	Label	Text	速度
④	NumericUpDown	(Name)	nudSpeed
		Increment	5
		Value	10
⑤	Label	Text	ゴミ
⑥	NumericUpDown	(Name)	nudGarbage
		DecimalPlaces	1
		Increment	0.5
⑦	Button	(Name)	btnAdd
		Text	登録
⑧	Label	(Name)	lblDisp
		Text	（任意）
⑨	Button	(Name)	btnStart
		Text	スタート

⑩	PictureBox	(Name)	pctCar
		Image	（ゴミ収集車.png）
		SizeMode	StretchImage
⑪	Timer	(Name)	tmrRun

フィールド		
gbg	GarbageTruck 型	GarbageTruck クラスのオブジェクトの参照を格納するための変数

コントロール	⑦	イベント	Click
GarbageTruck クラスのオブジェクトを生成し、フィールド変数 gbg へ格納する。(GarbageTruck クラスについては後述) ②と④と⑥に入力された値を、SetGarbageData()メソッドでそのオブジェクトへ設定する。 フィールド変数 gbg の Disp()メソッドを用いて現在の情報を⑧へ表示させる。			
コントロール	⑨	イベント	Click
⑩の表示位置を初期値に戻す。 ⑪の Interval を次の式を用いて設定する。「　210 - 速度 * 2　」 ⑪のタイマーを開始する。			
コントロール	⑪	イベント	Tick
フィールド変数 gbg の燃料を 1 減らす。 フィールド変数 gbg の CollectGarbage()メソッドを用いてゴミを増やす。 ⑩の表示位置を右に 1 ずらす。 フィールド変数 gbg の Disp()メソッドを用いて現在の情報を⑧へ表示させる。 もし燃料が 0 になった場合は⑪のタイマーを停止させる。			

クラス名	Car	アクセス修飾子	Public

フィールド

プロパティ名	型	内容
Fuel	Integer 型	燃料の情報を管理するためのプロパティ
Speed	Integer 型	速度の情報を管理するためのプロパティ

メソッド

プロシージャ	Sub	アクセス修飾子	Public
メソッド名	SetData	戻り値の型	（なし）
引数	Integer 型 , Integer 型		
第一引数で受け取った値をフィールドプロパティ Fuel へ格納する。 第二引数で受け取った値をフィールドプロパティ Speed へ格納する。			
プロシージャ	Function	アクセス修飾子	Public
メソッド名	Disp	戻り値の型	String 型
オーバーライド	Overridable		
引数	（なし）		
フィールドプロパティ Fuel および Speed をもとに表示文字列を返す。 表示例：「　燃料：100　速度：100　」			

クラス名	GarbageTruck	アクセス修飾子	Public
継承	Car		

フィールド

プロパティ名	型	内容
Garbage	Double 型	ゴミの情報を管理するためのプロパティ

メソッド

プロシージャ	Sub	アクセス修飾子	Public
メソッド名	SetGarbageData	戻り値の型	（なし）
引数	Integer 型 , Integer 型 , Double 型		
第一引数で受け取った値をフィールドプロパティ Fuel へ格納する。 第二引数で受け取った値をフィールドプロパティ Speed へ格納する。 第三引数で受け取った値をフィールドプロパティ Garbage へ格納する。			
プロシージャ	Function	アクセス修飾子	Public
メソッド名	Disp	戻り値の型	String 型
オーバーライド	Overrides		
引数	（なし）		
基本クラスの Disp()メソッドの結果とフィールドプロパティ Garbage をもとに表示文字列を返す 表示例：「　燃料：100　速度：100　ゴミ：100　」			
プロシージャ	Sub	アクセス修飾子	Public
メソッド名	CollectGarbage	戻り値の型	（なし）
引数	（なし）		
フィールドプロパティ Garbage に 0.5 加算する。			

解答例 Exercise06

【Form1.vb】

```
Public Class Form1

    Dim gbg As GarbageTruck

    Private Sub btnAdd_Click(sender As Object,
                             e As EventArgs) Handles btnAdd.Click

        gbg = New GarbageTruck()
        gbg.SetGarbageData(nudFuel.Value, nudSpeed.Value, nudGarbage.Value)

        lblDisp.Text = gbg.Disp()

    End Sub

    Private Sub btnStart_Click(sender As Object,
                               e As EventArgs) Handles btnStart.Click

        pctCar.Left = 12
        tmrRun.Interval = 210 - gbg.Speed * 2
        tmrRun.Start()

    End Sub

    Private Sub tmrRun_Tick(sender As Object,
                            e As EventArgs) Handles tmrRun.Tick

        gbg.Fuel -= 1
        gbg.CollectGarbage()
        pctCar.Left += 1
        lblDisp.Text = gbg.Disp()

        If gbg.Fuel <= 0 Then
            tmrRun.Stop()
        End If

    End Sub

End Class
```

【Car.vb】

```
Public Class Car

    Public Property Fuel As Integer
    Public Property Speed As Integer

    Public Sub SetData(tmpFuel As Integer, tmpSpeed As Integer)

        Fuel = tmpFuel
        Speed = tmpSpeed

    End Sub

    Public Overridable Function Disp() As String

        Return "  燃料:" & Fuel & "  速度:" & Speed

    End Function

End Class
```

【GarbageTruck.vb】

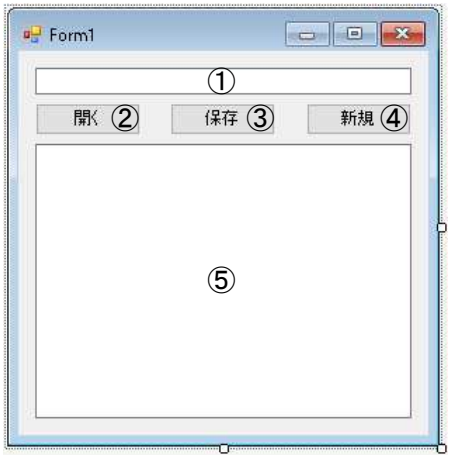
```
Public Class GarbageTruck
    Inherits Car

    Public Property Garbage As Double

    Public Sub SetGarbageData(tmpFuel As Integer,
                              tmpSpeed As Integer, tmpGarbage As Double)

        MyBase.SetData(tmpFuel, tmpSpeed)
        Garbage = tmpGarbage

    End Sub

    Public Overrides Function Disp() As String

        Return MyBase.Disp() & "  ゴミ:" & Garbage

    End Function

    Public Sub CollectGarbage()

        Garbage += 0.5

    End Sub

End Class
```


課題（ファイル処理）

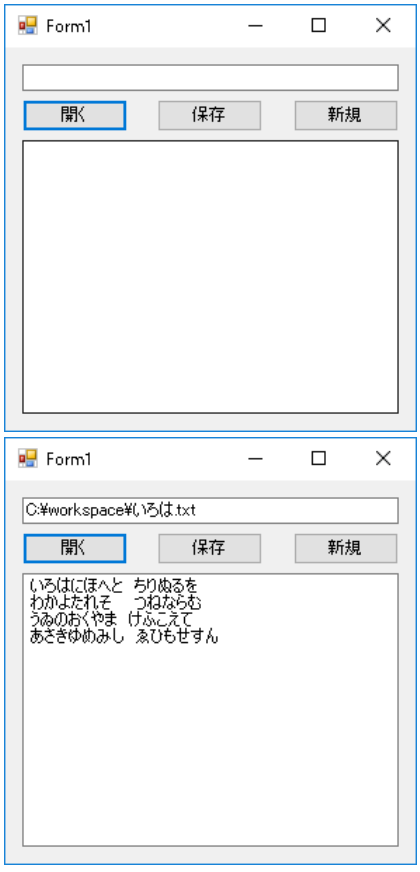
Windows では、C ドライブ直下へのファイル作成は制限されています。ファイル保存時は作業用のフォルダ内などを指定してください。

サンプル 7

【デザイン例】



【実行結果例】



番号	コントロール名	プロパティ名	プロパティ値
①	TextBox	(Name)	txtFile
②	Button	(Name)	btnOpen
		Text	開く
③	Button	(Name)	btnSave
		Text	保存
④	Button	(Name)	btnNew
		Text	新規
⑤	TextBox	(Name)	tbxMemo
		Multiline	True

コントロール	②	イベント	Click
OpenFileDialog クラスのオブジェクトを生成する。 OpenFileDialog クラスのオブジェクトの ShowDialog()メソッドで開くファイル名を選択させる。 ファイルが選択された場合 　選択されたファイルの名前（パス）を①へ表示する。 　①をもとに StreamReader クラスのオブジェクトを生成する。 　ファイルの内容を読込み、⑤へ表示させる。 　ファイルを閉じる。			
コントロール	③	イベント	Click
①が空欄の場合 　SaveFileDialog クラスのオブジェクトを生成する。 　SaveFileDialog クラスのオブジェクトの ShowDialog()メソッドで保存するファイル名を選択させる。 　ファイルが選択された場合 　　選択されたファイルの名前（パス）を①へ表示する。 　ファイルが選択されなかった場合（キャンセルが選択された場合） 　　このプロシージャを終了する。 ①をもとに StreamWriter クラスのオブジェクトを生成する。 　⑤の内容をファイルへ書込む。 　ファイルを閉じる。			
コントロール	④	イベント	Click
①と⑤の内容をクリアする。			

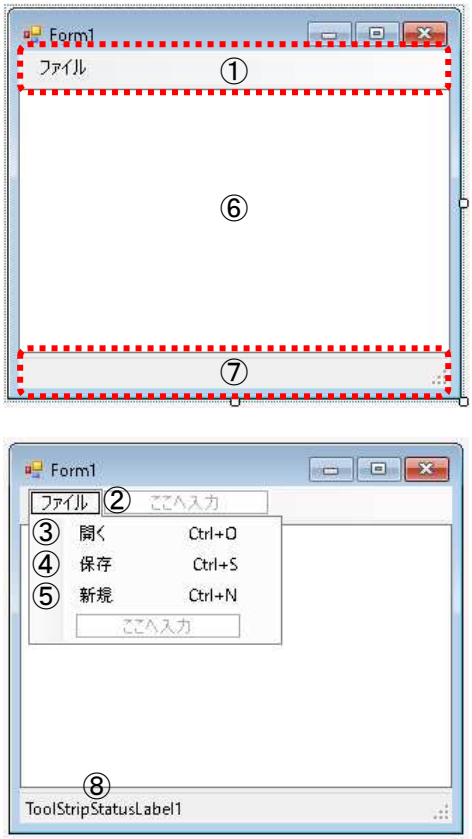
実装 ExerciseSample07

【Form1.vb】

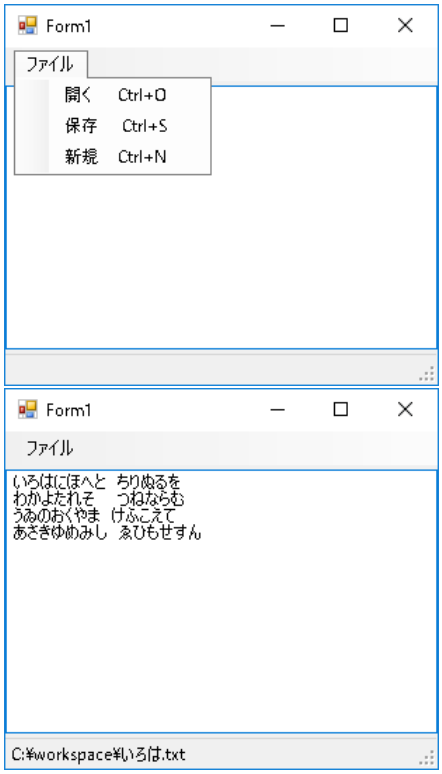
```
Imports System.IO

Public Class Form1

    Private Sub btnOpen_Click(sender As Object,
                              e As EventArgs) Handles btnOpen.Click

        Dim ofd As New OpenFileDialog()
        If ofd.ShowDialog() = DialogResult.OK Then

            txtFile.Text = ofd.FileName

            Dim sr As New StreamReader(txtFile.Text)
            tbxMemo.Text = sr.ReadToEnd()
            sr.Close()

        End If

    End Sub

    Private Sub btnSave_Click(sender As Object,
                              e As EventArgs) Handles btnSave.Click

        If txtFile.Text = "" Then

            Dim sfd As New SaveFileDialog()
            If sfd.ShowDialog() = DialogResult.OK Then

                txtFile.Text = sfd.FileName

            Else
                Exit Sub
            End If

        End If

        Dim sw As New StreamWriter(txtFile.Text)
        sw.WriteLine(tbxMemo.Text)
        sw.Close()

    End Sub

    Private Sub btnNew_Click(sender As Object,
                             e As EventArgs) Handles btnNew.Click

        txtFile.Text = ""
        tbxMemo.Text = ""

    End Sub

End Class
```

課題 7

【デザイン例】



【実行結果例】



番号	コントロール名	プロパティ名	プロパティ値
①	MenuStrip	ー	ー
②	ToolStripMenuItem	(Name)	mnuFile
		Text	ファイル
③	ToolStripMenuItem	(Name)	mnuOpen
		ShortcutKeys	Ctrl+O
		Text	開く
④	ToolStripMenuItem	(Name)	mnuSave
		ShortcutKeys	Ctrl+S
		Text	保存
⑤	ToolStripMenuItem	(Name)	mnuNew
		ShortcutKeys	Ctrl+N
		Text	新規
⑥	TextBox	(Name)	tbxMemo
		Multiline	True

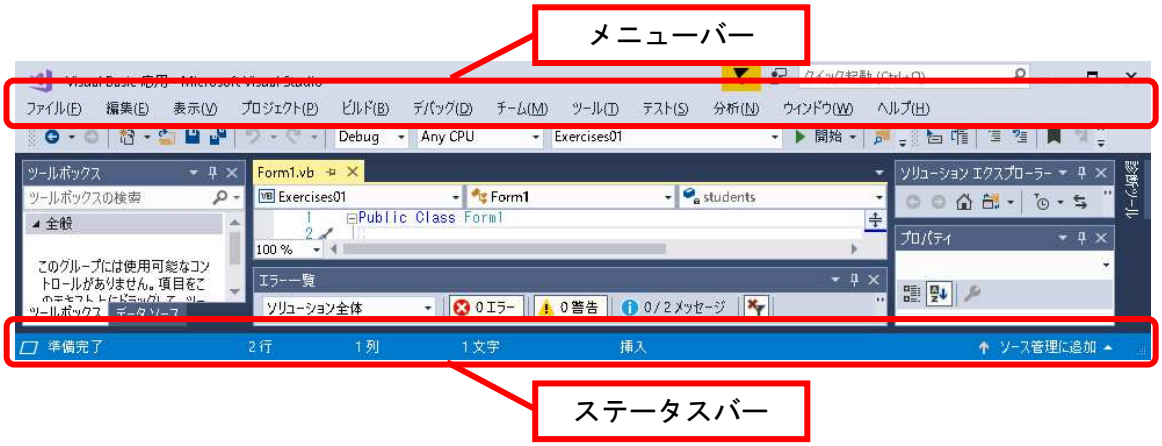
⑦	StatusStrip	ー	ー
⑧	ToolStripStatusLabel	(Name)	stsFile
		Text	（なし）

コントロール	③	イベント	Click
OpenFileDialog クラスのオブジェクトを生成する。 OpenFileDialog クラスのオブジェクトの ShowDialog()メソッドで開くファイル名を選択させる。 ファイルが選択された場合 　選択されたファイルの名前（パス）を⑧へ表示する。 　⑧をもとに StreamReader クラスのオブジェクトを生成する。 　ファイルの内容を読込み、⑥へ表示させる。 　ファイルを閉じる。			
コントロール	④	イベント	Click
⑧が空欄の場合 　SaveFileDialog クラスのオブジェクトを生成する。 　SaveFileDialog クラスのオブジェクトの ShowDialog()メソッドで保存するファイル名を選択させる。 　ファイルが選択された場合 　　選択されたファイルの名前（パス）を⑧へ表示する。 　ファイルが選択されなかった場合（キャンセルが選択された場合） 　　このプロシージャを終了する。 ⑧をもとに StreamWriter クラスのオブジェクトを生成する。 　⑥の内容をファイルへ書込む。 　ファイルを閉じる。			
コントロール	⑤	イベント	Click
⑥と⑧の内容をクリアする。			

メニューバーとステータスバー

メニューバーは、基本的に当該アプリケーションで行える操作が一覧になってまとまっているところです。ただし、頻繁に使う操作については毎回メニューをたどっていたのでは手間がかかってしまうので、通常はショートカットアイコンやショートカットキーが用意されていることが多いです。

ステータスバーはメッセージボックスなどで表示させるまでもない、補足事項を表示させるエリアです。Visual Studio の場合、ソースコードのエディタの時はカーソルの位置（行番号、列番号）が表示されています。フォームエディタの時はコントロールの座標（Location）や大きさ（Size）が表示されています。



何を表示させるか、どう使うかはアプリケーションの作成者次第です。メニューバーに関しては「ファイル」「編集」・・・といった、一般的なアプリケーションが多く採用しているような傾向を参考に作成するとよいでしょう。今まで作成してきたアプリケーションのように、メニューバーやステータスバーは必ず必要なものではありません。

今後、アプリケーションに触れるときは気にするようにしてみてください。

メニューバーの作成方法

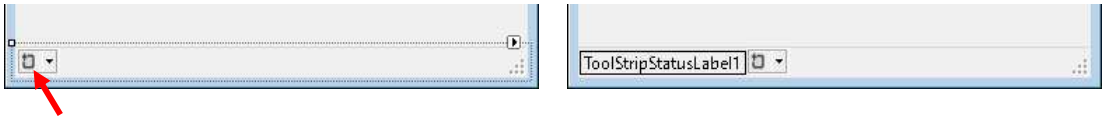
MenuStrip をデザイナのフォームへ配置したら、「ここへ入力」の部分をクリックし、メニューに表示させる文字列を入力します。

続いて、そのメニューの項目として表示させたい内容を同様の操作を繰り返して入力します。

メニューの項目を作成した後にその項目を選択すると、それぞれの項目（ToolStripMenuItem）のプロパティが設定できるようになっています。

ステータスバーの作成方法

ステータスバーも同様に、StatusStrip をデザイナのフォームへ配置したら、アイコン部分をクリックします。ToolStripStatusLabel が生成されるので、あとはプロパティウィンドウで設定可能です。さらにアイコンをクリックすることで要素を追加できます。



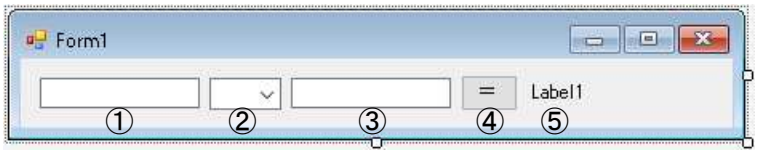
解答例 Exercise07

```
Imports System.IO

Public Class Form1

    Private Sub mnuOpen_Click(sender As Object,
                              e As EventArgs) Handles mnuOpen.Click

        Dim ofd As New OpenFileDialog()
        If ofd.ShowDialog() = DialogResult.OK Then

            stsFile.Text = ofd.FileName

            Dim sr As New StreamReader(stsFile.Text)
            tbxMemo.Text = sr.ReadToEnd()
            sr.Close()

        End If

    End Sub

    Private Sub mnuSave_Click(sender As Object,
                              e As EventArgs) Handles mnuSave.Click

        If stsFile.Text = "" Then

            Dim sfd As New SaveFileDialog()
            If sfd.ShowDialog() = DialogResult.OK Then

                stsFile.Text = sfd.FileName

            Else
                Exit Sub
            End If

        End If

        Dim sw As New StreamWriter(stsFile.Text)
        sw.WriteLine(tbxMemo.Text)
        sw.Close()

    End Sub

    Private Sub mnuNew_Click(sender As Object,
                             e As EventArgs) Handles mnuNew.Click

        stsFile.Text = ""
        tbxMemo.Text = ""

    End Sub

End Class
```

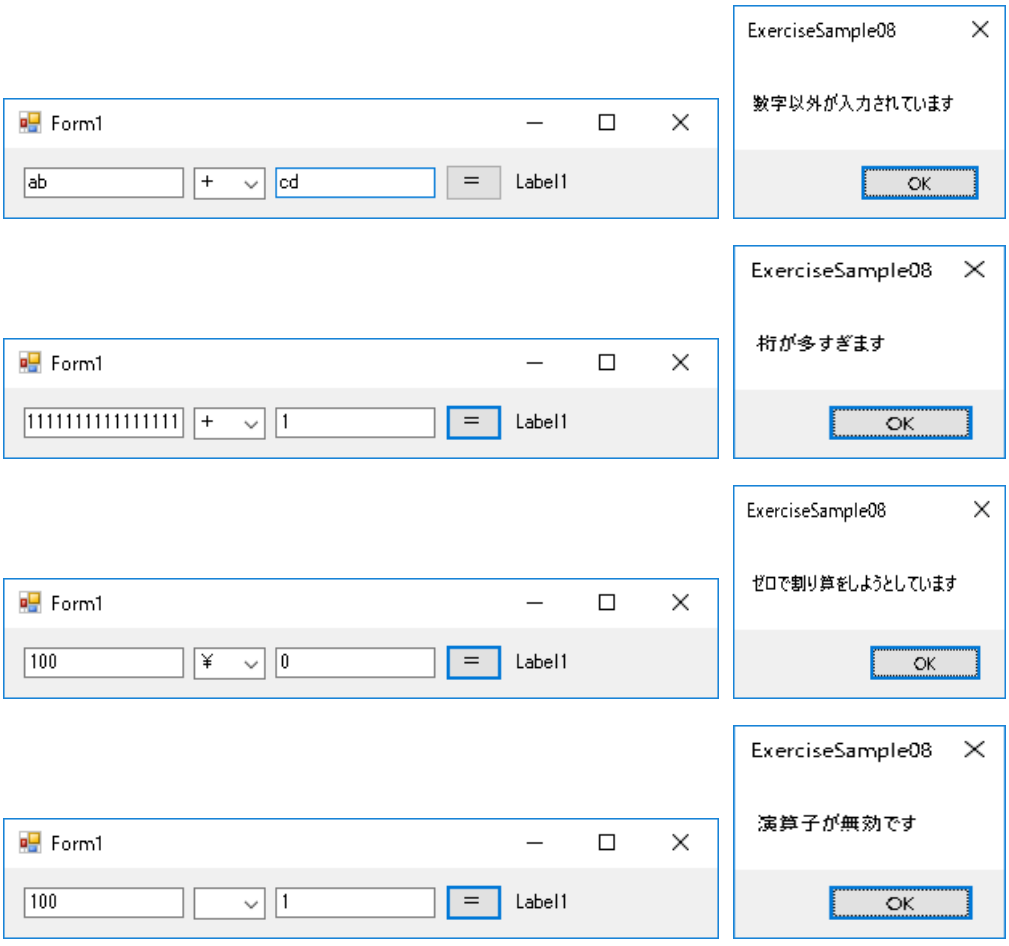

課題（例外処理）

サンプル 8

【デザイン例】



【実行結果例】



番号	コントロール名	プロパティ名	プロパティ値
①	TextBox	(Name)	txtNum1
②	ComboBox	(Name)	cmbOperator
		（選択候補）	＋ － ＊ ／ ￥ Mod ※プログラムで設定してもよい
③	TextBox	(Name)	txtNum2
④	Button	(Name)	btnCalc
		Text	＝
⑤	Label	(Name)	lblAns

コントロール	④	イベント	Click

以下の処理中に発生した例外は別記のパターン別に捕捉（Catch）し、メッセージを表示する。

①と③を数値（Double 型）に変換する。

②の選択内容によって①と③を演算する。選択が不正な場合は「演算子が無効です」とメッセージを表示する。

例外の捕捉	
例外クラス名	メッセージ（内容）
FormatException	数字以外が入力されています
OverflowException	桁が多すぎます
DiviedByZeroException	ゼロで割り算をしようとしています
Exception	想定外の例外が発生しました

コントロール	Form	イベント	Load

②へ以下の選択候補を追加する。

＋，－，＊，／，￥，Mod

※フォームデザイナー上で設定してもよい。

ゼロでの割り算

VB におけるゼロでの割り算は、サンプルの動作結果からもわかるように、次のように定義されています。

123 / 0	∞（無限大）
123 ¥ 0	例外（DivideByZeroException）発生
123 Mod 0	NaN（Not a Number）

それぞれの結果になる理由は数学的な知識が必要ですので詳しい解説は省略します。

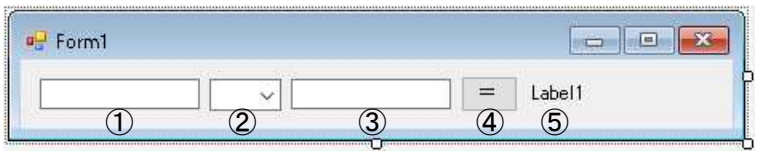
実装 **ExerciseSample08**

```
Public Class Form1

    Private Sub Form1_Load(sender As Object,
                           e As EventArgs) Handles MyBase.Load

        cmbOperator.Items.Add("＋")
        cmbOperator.Items.Add("－")
        cmbOperator.Items.Add("＊")
        cmbOperator.Items.Add("／")
        cmbOperator.Items.Add("￥")
        cmbOperator.Items.Add("Mod")

    End Sub

    Private Sub btnCalc_Click(sender As Object,
                              e As EventArgs) Handles btnCalc.Click

        Dim ans As Double = 0
        Dim str As String = ""

        Try
            Dim num1 As Double = Double.Parse(txtNum1.Text)
            Dim num2 As Double = Double.Parse(txtNum2.Text)

            Select Case (cmbOperator.Text)
                Case "＋" : ans = num1 + num2
                Case "－" : ans = num1 - num2
                Case "＊" : ans = num1 * num2
                Case "／" : ans = num1 / num2
                Case "￥" : ans = num1 ¥ num2
                Case "Mod" : ans = num1 Mod num2
                Case Else : str = "演算子が無効です"
            End Select

        Catch ex As FormatException
            str = "数字以外が入力されています"
        Catch ex As OverflowException
            str = "桁が多すぎます"
        Catch ex As DivideByZeroException
            str = "ゼロで割り算をしようとしています"
        Catch ex As Exception
            str = "想定外の例外が発生しました" & vbCrLf & ex.ToString()
        Finally
            If str = "" Then
                lblAns.Text = ans
            Else
                MsgBox(str)
            End If
        End Try

    End Sub

End Class
```

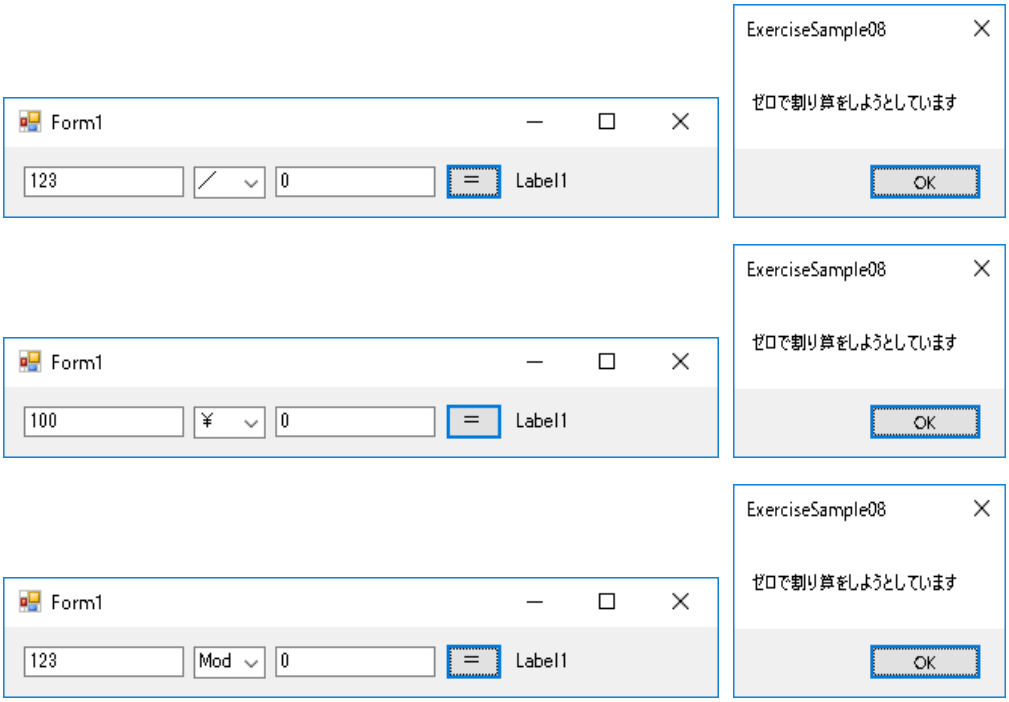

課題 8

先のサンプルに手を加えます。小数まで答えを求める割り算 / と、余りを求める割り算 Mod の場合も例外として処理をさせるように変更します。

【デザイン例】



【実行結果例】



番号	コントロール名	プロパティ名	プロパティ値
①	TextBox	(Name)	txtNum1
②	ComboBox	(Name)	cmbOperator
		（選択候補）	＋ － ＊ ／ ￥ Mod ※プログラムで設定してもよい
③	TextBox	(Name)	txtNum2
④	Button	(Name)	btnCalc
		Text	＝
⑤	Label	(Name)	lblAns

コントロール	④	イベント	Click

以下の処理中に発生した例外は別記のパターン別に捕捉（Catch）し、メッセージを表示する。

①と③を数値（Double 型）に変換する。

②の選択内容によって①と③を演算する。選択が不正な場合は「演算子が無効です」とメッセージを表示する。

ただし、②を ／ または Mod とした場合で、かつ③がゼロのとき、例外（DivideByZeroException）を発生させる。

例外の捕捉	
例外クラス名	メッセージ（内容）
FormatException	数字以外が入力されています
OverflowException	桁が多すぎます
DivideByZeroException	ゼロで割り算をしようとしています
Exception	想定外の例外が発生しました

コントロール	Form	イベント	Load

②へ以下の選択候補を追加する。

＋，－，＊，／，￥，Mod

※フォームデザイナー上で設定してもよい。

例外を意図的に発生させる

```
Dim ex As Exception = New Exception()
Throw ex
```

上記の 1 行目で発生させる例外を生成しています。例外の情報もオブジェクトです。オブジェクトを生成しただけではプログラムを強制終了させる効力はありません。

上記の 2 行目で例外を実際に発生させています。その後の挙動は、処理の中で自動的に発生した例外と同じです。

解答例 Exercise08

【Form1.vb】

```
Public Class Form1

    Private Sub Form1_Load(sender As Object,
                           e As EventArgs) Handles MyBase.Load

        cmbOperator.Items.Add("＋")
        cmbOperator.Items.Add("－")
        cmbOperator.Items.Add("＊")
        cmbOperator.Items.Add("／")
        cmbOperator.Items.Add("￥")
        cmbOperator.Items.Add("Mod")

    End Sub

    Private Sub btnCalc_Click(sender As Object,
                              e As EventArgs) Handles btnCalc.Click

        Dim ans As Double = 0
        Dim str As String = ""

        Try
            Dim num1 As Double = Double.Parse(txtNum1.Text)
            Dim num2 As Double = Double.Parse(txtNum2.Text)

            Select Case (cmbOperator.Text)
                Case "＋" : ans = num1 + num2
                Case "－" : ans = num1 - num2
                Case "＊" : ans = num1 * num2
                Case "／"
                    If num2 = 0 Then
                        Throw New DivideByZeroException()
                    End If
                    ans = num1 / num2
                Case "￥" : ans = num1 ¥ num2
                Case "Mod"
                    If num2 = 0 Then
                        Throw New DivideByZeroException()
                    End If
                    ans = num1 Mod num2
                Case Else : str = "演算子が無効です"
            End Select

        Catch ex As FormatException
            str = "数字以外が入力されています"
        Catch ex As OverflowException
            str = "桁が多すぎます"
        Catch ex As DivideByZeroException
            str = "ゼロで割り算をしようとしています"
        Catch ex As Exception
            str = "想定外の例外が発生しました" & vbCrLf & ex.ToString()
        Finally
            If str = "" Then
                lblAns.Text = ans
```

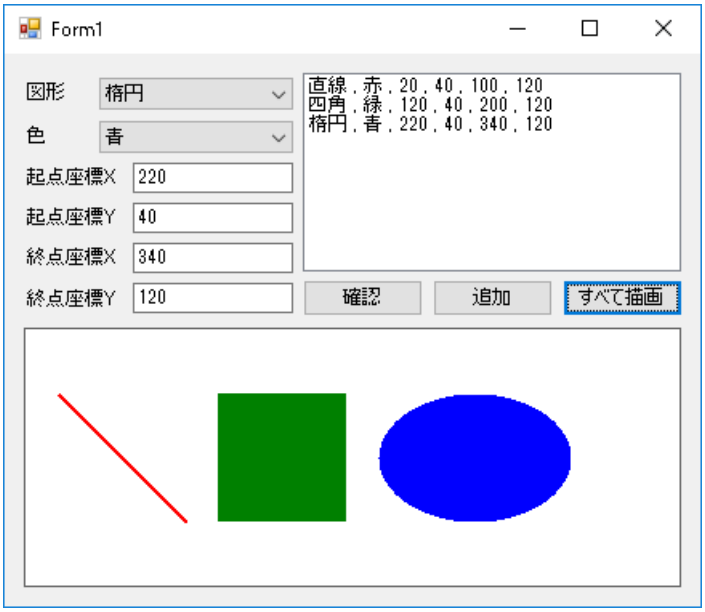
```
            Else
                MsgBox(str)
            End If

        End Try

    End Sub

End Class
```

最終課題

下記のような簡易的なお絵かきアプリを作成してください。

【デザイン例】

【実行結果例】



番号	コントロール名	プロパティ名	プロパティ値
①	Label	Text	図形
②	Label	Text	色
③	Label	Text	起点座標 X
④	Label	Text	起点座標 Y
⑤	Label	Text	終点座標 X
⑥	Label	Text	終点座標 Y
⑦	ComboBox	(Name)	cmbType
		DropDownStyle	DropDownList
		Items	四角 直線 楕円
⑧	ComboBox	(Name)	cmbColor
		DropDownStyle	DropDownList
		Items	赤 緑 青
⑨	TextBox	(Name)	txtX1
⑩	TextBox	(Name)	txtY1
⑪	TextBox	(Name)	txtX2
⑫	TextBox	(Name)	txtY2
⑬	ListBox	(Name)	lbxDisp
		Anchor	Top, Left, Right
⑭	Button	(Name)	btnChk
		Text	確認
⑮	Button	(Name)	btnAdd
		Text	追加
⑯	Button	(Name)	btnDraw
		Text	すべて描画
⑰	PictureBox	(Name)	picDraw
		Anchor	Top, Bottom, Left, Right
		BackColor	White
		BorderStyle	FixedSingle

フィールド		
shapeMax	Integer 型	図形の最大登録数
aryShape	Shape 型の配列	登録済みの図形
cnt	Integer 型	登録済みの図形の数
tmpShape	Shape 型	仮登録の図形

コントロール	Form	イベント	Load
⑦の選択状態を最初の内容とする ⑧の選択状態を最初の内容とする			
コントロール	⑰	イベント	MouseDown
イベントの MouseEventArgs 型の引数 e のプロパティ X の値を⑨に設定する イベントの MouseEventArgs 型の引数 e のプロパティ Y の値を⑩に設定する			
コントロール	⑰	イベント	MouseUp
イベントの MouseEventArgs 型の引数 e のプロパティ X の値を⑪に設定する イベントの MouseEventArgs 型の引数 e のプロパティ Y の値を⑫に設定する			
コントロール	⑭	イベント	Click

⑰の CreateGraphics()メソッドを呼び出すことで⑰の描画エリアを取得し、ローカル変数 gra へ格納する。

```
Dim gra As Graphics = ⑰.CreateGraphics( )
```

変数 gra の Clear()メソッドで描画エリアをクリアする。クリア時の色は白とする。

```
gra.Clear( Color.White )
```

⑦の選択状況によってそれぞれオブジェクトを生成する。引数には変数 gra を指定する。生成したオブジェクトはフィールド変数 tmpShape へ格納する。変数 tmpShape から SetData()メソッドを呼び出す。その際の引数は、⑨、⑩、⑪、⑫の値と⑧の文字列とする。

変数 tmpShape から DrawShape()メソッドを呼び出す。

コントロール	⑮	イベント	Click

図形の登録数が上限に達していなければ、図形を管理するフィールド変数（配列）へ追加する。また、⑬へ登録内容を表示させる。

コントロール	⑯	イベント	Click

現在登録されている図形をすべて描画する。

クラス名	Shape	アクセス修飾子	Public

フィールド

プロパティ名	型	内容
G	Graphics 型	描画エリアを管理する変数
C	Color 型	描画色（データ）を管理する変数
ColStr	String 型	描画色（文字列）を管理する変数
X1	Integer 型	図形描画のための X 座標その 1
Y1	Integer 型	図形描画のための Y 座標その 1
X2	Integer 型	図形描画のための X 座標その 2
Y2	Integer 型	図形描画のための Y 座標その 2

メソッド

プロシージャ	Sub	アクセス修飾子	Public
メソッド名	DrawShape	戻り値の型	（なし）
オーバーライド	Overridable		
引数	（なし）		
処理内容　なし			

プロシージャ	Function	アクセス修飾子	Public
メソッド名	GetInfo	戻り値の型	String 型
オーバーライド	Overridable		
引数	（なし）		
フィールドプロパティ ColStr、X1、Y1、X2、Y2、をもとに表示文字列を返す。 生成される文字列の例：「　，赤，10，20，30，40　」			

プロシージャ	Sub	アクセス修飾子	Public
メソッド名	SetData	戻り値の型	（なし）
引数	Integer 型，Integer 型，Integer 型，Integer 型，String 型		
第一引数で受け取った値をフィールドプロパティ X1 へ格納する。 第二引数で受け取った値をフィールドプロパティ Y1 へ格納する。 第三引数で受け取った値をフィールドプロパティ X2 へ格納する。 第四引数で受け取った値をフィールドプロパティ Y2 へ格納する。 第五引数で受け取った値をフィールドプロパティ ColStr へ格納する。			

第五引数の内容に応じて以下の処理を行う。

"赤"の場合はフィールドプロパティ C へ Color.Red を格納する。

"緑"の場合はフィールドプロパティ C へ Color.Green を格納する。

"青"の場合はフィールドプロパティ C へ Color.Blue を格納する。

上記以外の場合はフィールドプロパティ C へ Color.Black を格納する。

クラス名	ShapeLine	アクセス修飾子	Public
継承	Shape		

コンストラクタ

プロシージャ	Sub	アクセス修飾子	Public
引数	Graphics 型		
第一引数で受け取った値を基本クラスのフィールドプロパティ G へ格納する。			

メソッド

プロシージャ	Sub	アクセス修飾子	Public
メソッド名	DrawShape	戻り値の型	（なし）
オーバーライド	Overrides		
引数	（なし）		

基本クラスのフィールドプロパティ C をもとに Pen クラスのオブジェクトを生成する。

```
Dim p As Pen = New Pen( C )
```

基本クラスのフィールドプロパティ G の DrawLine()メソッドを呼び出し、図形（直線）を描画する。その際の引数は、先に作成した Pen クラスのオブジェクトと、フィールドプロパティ X1,Y1,X2,Y2 を指定する。

```
G.DrawLine(p, X1, Y1, X2, Y2)
```

プロシージャ	Function	アクセス修飾子	Public
メソッド名	GetInfo	戻り値の型	String 型
オーバーライド	Overrides		
引数	（なし）		

"直線"という文字列と、基本クラスの GetInfo()メソッドで得られる文字列を結合して返す。
生成される文字列の例：「　直線 , 赤 , 10 , 20 , 30 , 40　」

クラス名	ShapeRectangle	アクセス修飾子	Public
継承	Shape		

コンストラクタ

プロシージャ	Sub	アクセス修飾子	Public
引数	Graphics 型		
第一引数で受け取った値を基本クラスのフィールドプロパティ G へ格納する。			

メソッド

プロシージャ	Sub	アクセス修飾子	Public
メソッド名	DrawShape	戻り値の型	（なし）
オーバーライド	Overrides		
引数	（なし）		

基本クラスのフィールドプロパティ C をもとに SolidBrush クラスのオブジェクトを生成する。

```
Dim sb As SolidBrush = New SolidBrush( C )
```

基本クラスのフィールドプロパティ G の FillRectangle()メソッドを呼び出し、図形（四角）を描画する。その際の引数は、先に作成した SolidBrush クラスのオブジェクトと、フィールドプロパティ X1,Y1,X2,Y2 から算出した、描画の基準となる座標（X,Y）と図形の幅と高さを指定する。

```
G.FillRectangle(sb, X1, Y1, X2 - X1, Y2 - Y1)
```

プロシージャ	Function	アクセス修飾子	Public
メソッド名	GetInfo	戻り値の型	String 型
オーバーライド	Overrides		
引数	（なし）		

"四角"という文字列と、基本クラスの GetInfo()メソッドで得られる文字列を結合して返す。
生成される文字列の例：「　四角 , 赤 , 10 , 20 , 30 , 40　」

クラス名	ShapeEllipse	アクセス修飾子	Public
継承	Shape		

コンストラクタ

プロシージャ	Sub	アクセス修飾子	Public
引数	Graphics 型		
第一引数で受け取った値を基本クラスのフィールドプロパティ G へ格納する。			

メソッド

プロシージャ	Sub	アクセス修飾子	Public
メソッド名	DrawShape	戻り値の型	（なし）
オーバーライド	Overrides		
引数	（なし）		

基本クラスのフィールドプロパティ C をもとに SolidBrush クラスのオブジェクトを生成する。

```
Dim sb As SolidBrush = New SolidBrush( C )
```

基本クラスのフィールドプロパティ G の FillEllipse()メソッドを呼び出し、図形（楕円）を描画する。その際の引数は、先に作成した SolidBrush クラスのオブジェクトと、フィールドプロパティ X1,Y1,X2,Y2 から算出した、描画の基準となる座標（X,Y）と図形の幅と高さを指定する。

```
G.FillEllipse(sb, X1, Y1, X2 - X1, Y2 - Y1)
```

プロシージャ	Function	アクセス修飾子	Public
メソッド名	GetInfo	戻り値の型	String 型
オーバーライド	Overrides		
引数	（なし）		

"楕円"という文字列と、基本クラスの GetInfo()メソッドで得られる文字列を結合して返す。
生成される文字列の例：「　楕円 , 赤 , 10 , 20 , 30 , 40　」

機能拡張

内容の保存

アプリケーションを終了しても、前回の状態を再現できるようにファイルへ情報を保存できるようにしましょう。

保存する内容は、下記の情報を 1 行ずつテキストで書き込みます。読み込む場合は一行ずつ読み込み、6 行で 1 つの図形の情報とします。

・図形名
・色
・座標 X1
・座標 Y1
・座標 X2
・座標 Y2

図形情報の管理

現在は追加しかできません。変更や削除をできるようにしましょう。

図形の確認

現在は描画ボタンを押すまで図形の大きさや位置を確認できません。図形の起点と終点を決める操作中に確認できるようにしましょう。

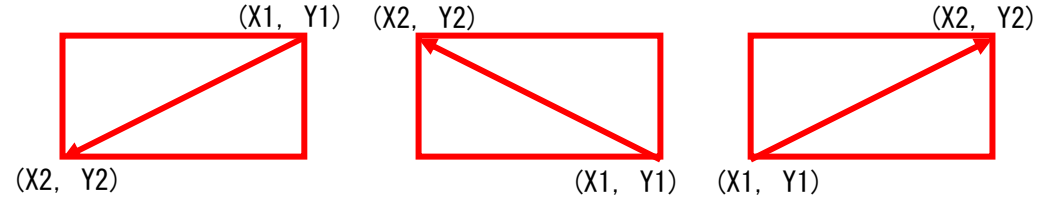
それには、MouseMove イベントを利用します。MouseMove イベントはマウスが動いたときに発生するイベントです。MouseDown イベントや MouseUp イベントと同様、引数からマウス操作時のカーソルの座標が取得できますので、それを利用して四角形の枠を書きましょう。

ただし、MouseMove イベントはボタンのクリック状態に関係なく発生します。今回はボタンが押されている場合（起点選択後）にだけ選択中の座標を表示させたいので、ボタンの押下状態を取得する必要があります。

また、MouseUp イベント発生時は図形が決定されるタイミングでもあります。現在は確認ボタンで図形を描画していますが、MouseUp イベント発生時に自動的に、現在調整中の図形と登録済みの図形の両方が描画されるようにしましょう。

座標指定の拡張

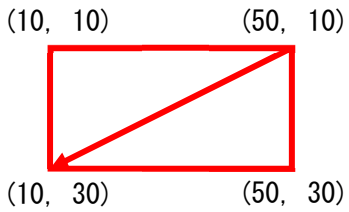
現在は図形の起点が左上に、終点が右下にないと、座標がマイナスになってしまって図形が描画されません。座標の指定位置のパターンは現在動作可能な左上から右下へのマウスの動きのほかに、下記の 3 パターンが考えられます。どのようなパターンでも図形が描画できるようにしましょう。



ヒント

図形を描画するには、左上の座標と右下の座標が必要です。（右下の座標が必要なのは高さと幅を算出するため）

左上の座標はどのパターンであっても起点と終点のどちらか座標の数値が少ないほうをとり、右下の座標はどのパターンであっても起点と終点のどちらか座標の数値が大きいほうを取ります。



解答例（機能拡張なし）

【Form1.vb】

```
Public Class Form1

    Dim shapeMax As Integer = 10        '図形の最大登録数
    Dim aryShape(shapeMax - 1) As Shape '登録済みの図形
    Dim cnt As Integer = 0              '登録済みの図形の個数
    Dim tmpShape As Shape               '仮登録の図形

    Private Sub Form1_Load(sender As Object,
                           e As EventArgs) Handles MyBase.Load
        cmbType.SelectedIndex = 0       '図形描画種別の選択を最初の内容とする
        cmbColor.SelectedIndex = 0      '図形描画色の選択を最初の内容とする
    End Sub

    'マウスのボタンを押したとき
    Private Sub picDraw_MouseDown(sender As Object,
                                  e As MouseEventArgs) Handles picDraw.MouseDown
        txtX1.Text = e.X
        txtY1.Text = e.Y
    End Sub

    'マウスのボタンを離したとき
    Private Sub picDraw_MouseUp(sender As Object,
                                e As MouseEventArgs) Handles picDraw.MouseUp
        txtX2.Text = e.X
        txtY2.Text = e.Y
    End Sub

    '「確認」ボタンを押したとき
    Private Sub btnChk_Click(sender As Object,
                             e As EventArgs) Handles btnChk.Click

        Dim gra As Graphics = picDraw.CreateGraphics   '描画対象

        gra.Clear(Color.White)          '描画エリアをクリア

        Select Case cmbType.Text
            Case "直線"
                tmpShape = New ShapeLine(gra)

            Case "四角"
                tmpShape = New ShapeRectangle(gra)

            Case "楕円"
                tmpShape = New ShapeEllipse(gra)

        End Select

        tmpShape.SetData(Integer.Parse(txtX1.Text), Integer.Parse(txtY1.Text),
                         Integer.Parse(txtX2.Text), Integer.Parse(txtY2.Text),
                         cmbColor.Text)
        tmpShape.DrawShape()

    End Sub
```

```
    '「追加」ボタンを押したとき
    Private Sub btnAdd_Click(sender As Object,
                             e As EventArgs) Handles btnAdd.Click

        If cnt < shapeMax Then

            aryShape(cnt) = tmpShape
            cnt += 1

            lbxDisp.Items.Add(tmpShape.GetInfo())

        End If

    End Sub

    '「すべて描画」ボタンを押したとき
    Private Sub btnDraw_Click(sender As Object,
                              e As EventArgs) Handles btnDraw.Click

        For num As Integer = 0 To cnt - 1

            aryShape(num).DrawShape()

        Next

    End Sub

End Class
```

【Shape.vb】

```
Public Class Shape

    Public Property G As Graphics
    Public Property C As Color
    Public Property ColStr As String

    Public Property X1 As Integer
    Public Property Y1 As Integer
    Public Property X2 As Integer
    Public Property Y2 As Integer

    Public Overridable Sub DrawShape()

    End Sub

    Public Overridable Function GetInfo() As String

        Return "," & ColStr &
               "," & X1 & "," & Y1 & "," & X2 & "," & Y2

    End Function

    Public Sub SetData(X1 As Integer, Y1 As Integer,
                X2 As Integer, Y2 As Integer, colStr As String)

        MyClass.X1 = X1
```

```
        MyClass.Y1 = Y1
        MyClass.X2 = X2
        MyClass.Y2 = Y2

        MyClass.ColStr = colStr

        Select Case colStr
            Case "赤"
                C = Color.Red
            Case "緑"
                C = Color.Green
            Case "青"
                C = Color.Blue
            Case Else
                C = Color.Black
        End Select

    End Sub

End Class
```

MyClass は自分のクラス内のフィールドを指すキーワードです。
引数名とフィールドのプロパティ名が同じなので区別をするために記述しています。
名前が異なっていれば不要です。

【ShapeLine.vb】

```
Public Class ShapeLine
    Inherits Shape

    Public Sub New(g As Graphics)
        MyBase.G = g
    End Sub

    Public Overrides Sub DrawShape()

        Dim p As Pen = New Pen(C, 2)     '第二引数は太さ
        G.DrawLine(p, X1, Y1, X2, Y2)

    End Sub

    Public Overrides Function GetInfo() As String

        Return "直線" & MyBase.GetInfo()

    End Function

End Class
```

【ShapeRectangle.vb】

```
Public Class ShapeRectangle
    Inherits Shape

    Public Sub New(g As Graphics)
        MyBase.G = g
    End Sub

    Public Overrides Sub DrawShape()

        Dim sb As SolidBrush = New SolidBrush(C)
```

```
            G.FillRectangle(sb, X1, Y1, X2 - X1, Y2 - Y1)

        End Sub

        Public Overrides Function GetInfo() As String

            Return "四角" & MyBase.GetInfo()

        End Function

    End Class
```

【ShapeEllipse.vb】

```
    Public Class ShapeEllipse
        Inherits Shape

        Public Sub New(g As Graphics)
            MyBase.G = g
        End Sub

        Public Overrides Sub DrawShape()

            Dim sb As SolidBrush = New SolidBrush(C)
            G.FillEllipse(sb, X1, Y1, X2 - X1, Y2 - Y1)

        End Sub

        Public Overrides Function GetInfo() As String

            Return "楕円" & MyBase.GetInfo()

        End Function

    End Class
```

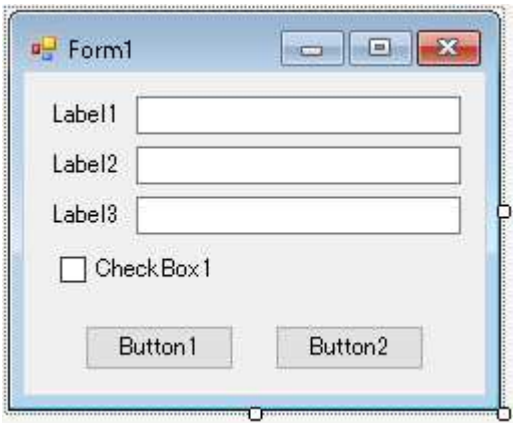
APPENDIX

フォームアプリケーションを作成するにあたって、普段何気なく使っている機能でも、いざ自分が実装しようとすると、どのように設定しているかわからないものがあります。そのなかでも簡単に設定できて便利な機能を取り上げます。

TabIndex

下の画面のように、フォーム上に入力項目やボタンなどのコントロール（部品）が複数ある場合、連続で入力したいと思います。入力のたびにマウスで次の項目を選択するのは効率が悪いので、できればキーボード上の操作で次の項目へ移動させたいです。

キーボードを操作した時、入力を受け付ける状態になっているコントロールの状態を「フォーカス」と言います。例えば、テキストボックスをクリックし、入力を受け付ける状態になることを「フォーカスを受け取る」「フォーカスを得る」と言ったり、他のコントロールへフォーカスが移動することを「フォーカスを失う」と言ったりします。



基本的に、Tab キーを押すと、次の部品へフォーカスが移動します。Web ページでも同様な結果を得られることが多いので、ご存知の方は少なくないと思います。

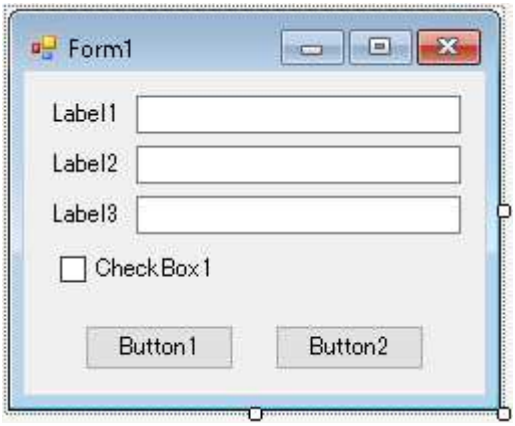
VB では、Tab キーを押したときフォーカスが移動する順番は、フォーム上の配置場所ではなく、作成時に配置した順番です。例えば、CheckBox1→TextBox1 の順番で配置した場合、CheckBox1 の操作後に Tab キーを押すと、TextBox1 の入力エリアにカーソルが移動します。

あとでコントロールを間に追加したくなった場合はどうすればよいでしょうか。レイアウト上は他のコントロールをずらせば間に配置できますが、Tab の移動の順番は最後になってしまいます。

Tab を押したときの移動の順番は TabIndex プロパティで決まります。TabIndex プロパティはフォーム上へコントロールを配置した時に自動的に設定されます。コントロールの種類は関係なく、配置するたびに 0 から順に自動的に 1 ずつ加算されて設定されます。Label や Panel のようにフォーカスを受け取らない部品もありますが、TabIndex は設定されます。

下の表はサンプルフォームを作成するときに、配置したコントロールの順番と TabIndex の初期値です。

配置した順番	コントロール	TabIndex
1	Label1	0
2	TextBox1	1
3	Label2	2
4	TextBox2	3
5	Label3	4
6	TextBox3	5
7	CheckBox1	6
8	Button1	7
9	Button2	8



この場合、フォーム起動時に最初にフォーカスが設定されるのは TextBox1 です。その状態から Tab キーを押すと TextBox2 へフォーカスが移動します。Label はフォーカスを受け取らないので無視されます。Button2 の次はまた TextBox1 にフォーカスが移ります。

CheckBox はフォーカスが設定されている状態でスペースキーを押すことで、マウスでクリックしたときのようにチェックを ON/OFF できます。

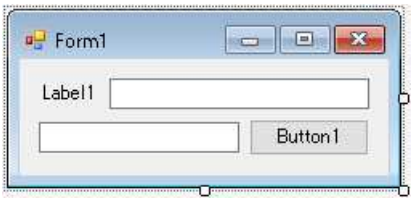
コントロールを配置した後でも TabIndex を直接指定することで、作成者が意図する順番にフォーカスを移動させることができます。

また、TabStop プロパティを False にすることで、フォーカスを受け取れるコントロールでもフォーカスの移動の対象外に設定できます。Tab キーを押してもフォーカス設定の対象にならないだけで、マウスでの操作やキーボードでの入力は可能です。

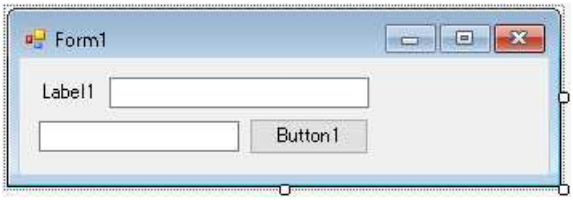
また、Shift キーを押しながら Tab キーを押すと、逆順にフォーカスを移動できます。

Anchor

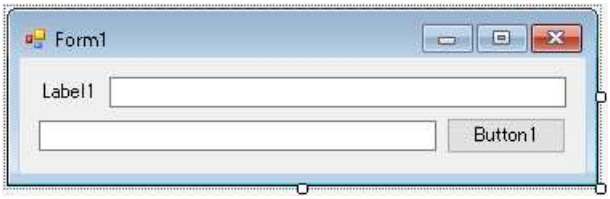
下の画面のようなフォームを作成したとします。



ラベルやテキストボックス、ボタンといったコントロール（部品）をただ配置しただけだと、フォームの大きさを変更したとき、次の画面のようになってしまいます。

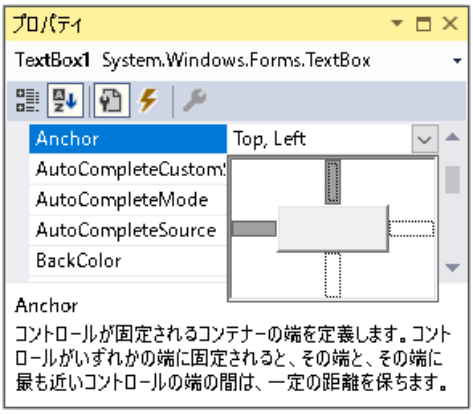


そこで、フォームの大きさを変更した際に、下の画面のようにフォーム上のコントロールの大きさもそれに応じて変更させたいと思います。

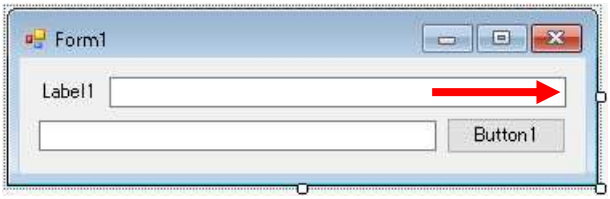


まずは上段の TextBox の設定を変更します。フォームのリサイズに対応してコントロールの大きさを自動的に変更させるには Anchor プロパティを設定します。

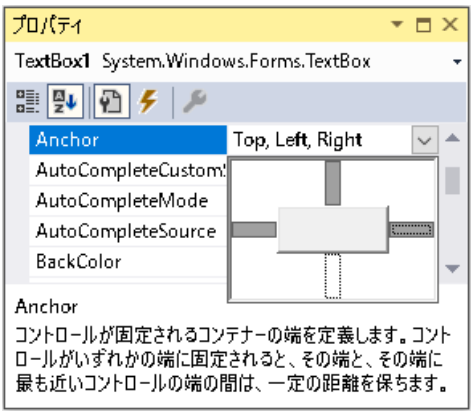
初期状態では下の画面のようになっています。



これは、フォームの大きさが変更されたときに、TextBox1 が上と左に固定されていることを表しています。今回はフォームの幅を広げた際に、テキストボックスの右側をフォームの右側と連動させて、一緒に動かしたいです。

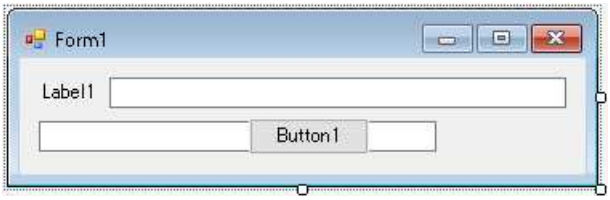


その場合は、Anchor に Right を追加します。テキストを編集しても変更可能ですが、灰色の部分をクリックするほうが容易に変更できます。

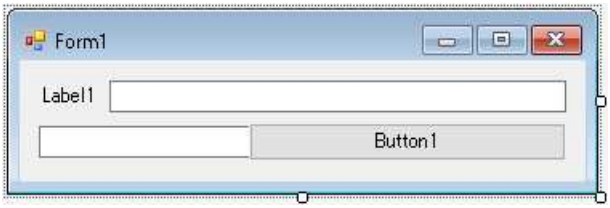


Anchor は日本語では船などを止めておく錨（いかり）の意味です。テキストボックスの右側をフォームから離れないように設定しているようなイメージです。

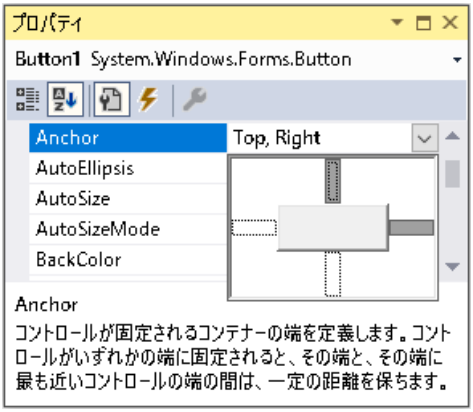
では、下段のテキストボックスも同様に、Anchor に Right を追加してみます。
そこで、フォームの大きさを変更すると、次の画面のようになってしまいます。



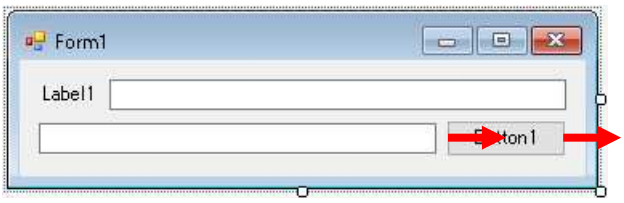
これは、ボタンの Anchor を変更していないため、その場にとどまったからです。ボタンの Anchor にも Right を追加し、もう一度確認してみます。



現在のボタンの Anchor は Top, Left, Right です。これでは、ボタンの左側が固定されたまま、右側がフォームに追随してしまいます。ボタンの大きさを変更しないで移動したように見せる場合は、Anchor の設定から Left を除きます。



そうすると、ボタンは左側の固定が外れ、右側が固定されるので、次の画面のようになります。



Visual Basic 応用講座テキスト　　P-060

初版発行日：　2014 年 03 月 30 日
最終更新日：　2021 年 03 月 02 日
著　　作：　株式会社シンクスバンク
発 行 者：　株式会社シンクスバンク