**MDIアプリケーション開発の準備**

　今回は、Windowsアプリケーションの基本部分を作成する。具体的には、MDI型のWindowsアプリケーションを作成し、メニュー・バーやステータス・バーの追加、子ウィンドウを1つしか起動できないようにする処理などの、MDIアプリケーションとして基礎的な機能を実装する。

　今回作成するVS.NETプロジェクトは、次のリンクよりダウンロードできる。

* VS.NET 2003プロジェクト（[C#版](https://www.atmarkit.co.jp/fdotnet/winexp/winexp01/cs2003_01.zip)／[VB.NET版](https://www.atmarkit.co.jp/fdotnet/winexp/winexp01/vb2003_01.zip)）
* VS.NET 2002プロジェクト（[C#版](https://www.atmarkit.co.jp/fdotnet/winexp/winexp01/cs2002_01.zip)／[VB.NET版](https://www.atmarkit.co.jp/fdotnet/winexp/winexp01/vb2002_01.zip)）

　まずは、アプリケーション開発のフォルダ構成を決めておこう。

**■MDIアプリケーション開発のフォルダ構成**

　本稿では、次の図のようにフォルダを構成する。それぞれのフォルダは、Windows標準のエクスプローラなどを使って手動で作成すればよい。フォルダの意味については、図の下部にある説明を読んでほしい。

|  |
| --- |
|  |
| **MDIアプリケーションのフォルダ構成** |
| 本稿では、このようなフォルダ構成でアプリケーションの開発を進める。それぞれのフォルダは、Windows標準のエクスプローラなどを使って手動で作成すればよい。 |
| |  |  |  | | --- | --- | --- | |  |  | 「Insider.NET」フォルダ。アプリケーションのルート・フォルダ。通常は、このフォルダの名前は、会社名にすればよい。 | |  |  | 「Kokyaku.NET」フォルダ。アプリケーション名のフォルダ。本稿のアプリケーション名は、「Kokyaku.NET」とする。 | |  |  | 「bin」フォルダ。「bin」フォルダ内には「Debug」フォルダと「Release」フォルダがあり、「Debug」フォルダ内にはデバッグ版のプログラムが出力され、「Release」フォルダ内にはリリース版のプログラムが出力される。 | |  |  | 「doc」フォルダ。アプリケーションに関連するドキュメントを格納するためのフォルダ。 | |  |  | 「src」フォルダ。このフォルダに、アプリケーションのプロジェクトと、それに含まれるソース・ファイルを置く。 | |

　上図の構成でアプリケーション開発用のフォルダを準備できたら、次にVS.NETでWindowsアプリケーションのプロジェクトを作成しよう。

**■ソリューションの作成**

　VS.NETのプロジェクトには、それを管理するためのコンテナとなるVS.NETソリューションが必要となる。よって、VS.NETソリューションを先ほどのフォルダ構成の「src」フォルダの直下に作成する。具体的には、次の画面のようにして、ソリューションの新規作成を行う。

|  |
| --- |
|  |
| **VS.NETソリューションの作成** |
| VS.NETのIDEにあるメニュー・バーから［ファイル］－［新規作成］－［空のソリューション］を選択すると、［新しいプロジェクト］ダイアログが表示される。なお、このダイアログで［新しいソリューション名］の入力項目が表示されていない場合は、左下にある［詳細／縮小］ボタンをクリックすれば表示されるようになる。 |
| |  |  |  | | --- | --- | --- | |  |  | ［プロジェクトの種類］から「Visual Studio ソリューション」を選択する（自動的に選択されているはず）。 | |  |  | ［テンプレート］から［空のソリューション］を選択する（これも自動選択されているはず）。 | |  |  | ［プロジェクト名］にアプリケーション名（本稿では「Kokyaku.NET」）を入力する。 | |  |  | ［場所］に先ほど作成した「src」フォルダ（本稿の例では「C:\Insider.NET\Kokyaku.NET\src」）を入力する。 | |  |  | ソリューションの作成場所が表されているが、「src」フォルダの直下ではなく、「C:\Insider.NET\Kokyaku.NET\src\Kokyaku.NET」というフォルダになっている。このようにVS.NETでソリューションを作成すると、ソリューション名のフォルダを自動作成してしまうので、ソリューション作成後にこのフォルダの中身を「src」フォルダへと移動させる（詳細後述）。 | |  |  | ［OK］ボタンをクリックすると、空のソリューションが作成される。 | |

　上の画面の手順によりソリューションが作成されるが、実際に作成されたVS.NETソリューションのファイルを見ると、「src」フォルダの直下ではなく、「C:\Insider.NET\Kokyaku.NET\src**\Kokyaku.NET**」というフォルダの中に作成されている。これは、VS.NETが自動的にソリューション名のフォルダを作成してしまうためだ。

　そこで、作成されたソリューションを手動で「src」フォルダの直下に移動させることにしよう。フォルダの移動は、Windowsでの通常のファイル操作と同じように、Windowsのエクスプローラなどを使って行えばよい。次の画面は、実際にエクスプローラで移動しているところだ。

|  |
| --- |
|  |
| **ソリューション・ファイルの移動** |
| VS.NETでソリューションを作成すると、ソリューション名のフォルダの中にソリューション・ファイルが作成されるので、Windowsのエクスプローラなどを使って手動で、それらのファイルを「src」フォルダの直下に移動する。 |
| |  |  |  | | --- | --- | --- | |  |  | ソリューション名のフォルダ（本稿の例では「Kokyaku.NET」）を選択する。 | |  |  | ソリューション・ファイル（本稿の例では「Kokyaku.NET.sln」と「Kokyaku.NET.suo」）をドラッグ＆ドロップで「src」フォルダに移動する。 | |  |  | 移動した後のソリューション名のフォルダは不要なので削除する。 | |

　以上でソリューションの作成は完了だ。後はこのソリューションに、アプリケーションで必要となるプロジェクトを作成し、追加していくことになる。

**■ソリューションへのプロジェクトの追加**

　それでは実際に、ソリューションへプロジェクトを追加してみよう。ここでは、MDIアプリケーションのメイン・ウィンドウのプロジェクトを追加する。なお、今後の連載で、この「プロジェクトの追加」は何度も出てくるが、追加方法の説明は二度と繰り返さないので、ここで覚えてほしい。

　まず先ほどのソリューション・ファイル「Kokyaku.NET.sln」をダブルクリックして実行しよう。すると、VS.NETのIDEがそのソリューションを開いて起動するはずだ。次に起動したIDEの［ソリューション エクスプローラ］にある「ソリューション 'Kokyaku.NET'」を右クリックしてコンテキスト・メニューを表示し、そのメニューから［追加］－［新しいプロジェクト］を選択する。これにより、［新しいプロジェクトの追加］ダイアログが表示される。

|  |
| --- |
|  |
| **［新しいプロジェクトの追加］ダイアログによるプロジェクトの追加** |
| ［新しいプロジェクトの追加］ダイアログを使って、新たなプロジェクト（この例では「Kokyaku」）を追加する。 |
| |  |  |  | | --- | --- | --- | |  |  | VS.NET IDEの［ソリューション エクスプローラ］にある「ソリューション 'Kokyaku.NET'」を右クリックしてコンテキスト・メニューを表示し、そのメニューから［追加］－［新しいプロジェクト］を選択すると、［新しいプロジェクトの追加］ダイアログが表示される。このダイアログから新たなプロジェクト（本稿の例では「メイン・ウィンドウのプロジェクト」）を追加できる。 | |  |  | ［プロジェクトの種類］から「Visual C# プロジェクト」を選択する。 | |  |  | ［テンプレート］から［Windows アプリケーション］を選択する。 | |  |  | ［プロジェクト名］にプロジェクトの名前（本稿では「Kokyaku」）を入力する。 | |  |  | ［場所］に先ほど作成した「src」フォルダ（本稿の例では「C:\Insider.NET\Kokyaku.NET\src」）を入力する。 | |  |  | プロジェクトの作成場所（この例では「C:\Insider.NET\Kokyaku.NET\src\Kokyaku」）が表される。 | |  |  | ［OK］ボタンをクリックすると、プロジェクトが新規作成される。 | |

　上の画面のように、本稿では「Kokyaku」というプロジェクトを作成した。次に、このプロジェクトから生成されるプログラム（実行ファイル）が、先ほどのフォルダ構成で示した「bin」フォルダ内に出力されるように設定しよう。

**■プログラム出力先の設定**

　プログラムの出力先は、プロジェクトの［プロパティ ページ］ダイアログから設定できる。［プロパティ ページ］ダイアログを表示するには、VS.NET IDEの［ソリューション エクスプローラ］にある「Kokyaku」プロジェクトを右クリックしてコンテキスト・メニューを表示し、そのメニューから［プロパティ］を選択すればよい。なお、プログラム出力先の変更は、プロジェクトを新規作成するたびに行わなければならない。

　［プロパティ ページ］ダイアログは、次の画面のように設定すればよい。

|  |
| --- |
|  |
| **［プロパティ ページ］ダイアログによるプログラム出力先の設定** |
| プログラムの出力先として、先ほど作成した「bin」フォルダを指定する。この際、ソリューション構成が「Debug」モードのときのプログラム出力先は「bin」フォルダ内の「Debug」フォルダを指定し、さらに「Release」モードのときの出力先は「Release」フォルダを指定する（両方とも設定する必要がある）。なお、VB.NETの場合の［プロパティ ページ］ダイアログは見た目が少し違うが、設定内容は同じである。 |
| |  |  |  | | --- | --- | --- | |  |  | VS.NET IDEの［ソリューション エクスプローラ］にある「Kokyaku」プロジェクトを右クリックしてコンテキスト・メニューを表示し、そのメニューから［プロパティ］を選択すると、［プロパティ ページ］ダイアログ（この例では「Kokyaku プロパティ ページ」）が表示される。このダイアログでプログラムの出力先を設定・変更できる。 | |  |  | ツリー表示の項目の中から、［構成プロパティ］－［ビルド］を選択する。 | |  |  | まずはソリューション構成が「Debug」モードのときのプログラム出力先を設定するために、構成のコンボボックスから「Debug」を選択する。 | |  |  | プログラムの出力先として、［出力パス］の値を「..\..\bin\Debug」に変更する（C#の場合、デフォルトでは「bin\Debug」が設定されている。VB.NETでは「bin」となっている）。この設定変更により、デフォルトでは「Insider.NET\Kokyaku.NET\**src\Kokyaku**\bin\Debug」に出力されるプログラムが、「Insider.NET\Kokyaku.NET\bin\Debug」に出力されるようになる。入力値にある「..」は1階層上がることを意味し、「..\..」で2階層上がることになる。つまり、プロジェクト・フォルダから「src\Kokyaku」分の2階層上がった「Insider.NET\Kokyaku.NET」フォルダの下にある「bin\Debug」フォルダに出力されることになる。 | |  |  | 次に「Release」モードのときのプログラム出力先を設定するために、構成のコンボボックスから「Release」を選択する。 | |  |  | プログラムの出力先として、［出力パス］の値を「..\..\bin\Release」に変更する（C#の場合、デフォルトでは「bin\Release」が設定されている。VB.NETでは「bin」となっている）。この設定変更により、「Insider.NET\Kokyaku.NET\bin\Release」にプログラムが出力されるようになる。 | |  |  | ［OK］ボタンを押すと、プログラム出力先の設定が保存される。 | |

　上の画面のようにして、ソリューション構成が「Debug」モードのときのプログラム出力先を「Insider.NET\Kokyaku.NET\bin\Debug」に設定し、「Release」モードのときのプログラム出力先を「Insider.NET\Kokyaku.NET\bin\Release」に設定する。以上で、MDIアプリケーション開発の準備は整った。

　次に、MDIアプリケーションのコーディング作業に入っていこう。

**MDIアプリケーションの実装**

　先ほどWindowsアプリケーションの「Kokyaku」プロジェクトを作成したが、これがMDIアプリケーションのメイン・ウィンドウとなる。プロジェクトの新規作成によりひな形として自動生成されたWindowsフォームを、MDIアプリケーションのメイン・ウィンドウに変更するには、［IsMdiContainer］プロパティを「True」に設定変更するだけだ。

　さらにここで、メイン・ウィンドウのファイル名を「MainWindow.cs」に変更し、次のプロパティ設定を行おう。

* ［(Name)］プロパティ：「**MainWindow**」
* ［Text］プロパティ：「**顧客.NET**」
* ［Size］プロパティ：「**580, 480**」

　次の画面は、これらの設定を実際に行っているところである。

|  |
| --- |
|  |
| **MDIアプリケーションのメイン・ウィンドウのプロパティ設定** |
| VS.NETが生成したWindowsフォームのひな形をMDIアプリケーションのメイン・ウィンドウに変更するためにプロパティ設定を行っているところ。 |
| |  |  |  | | --- | --- | --- | |  |  | Windowsフォームのファイル名を「Form1.cs」から「MainWindow.cs」に変更する。 | |  |  | ［プロパティ］ウィンドウで、［IsMdiContainer］プロパティを「True」に設定変更すると、MDIアプリケーションのメイン・ウィンドウとなる。 | |  |  | ［(Name)］プロパティの値を「MainWindow」に変更する。またこの画面上には表示されていないが、［Text］プロパティを「顧客.NET」に、［Size］プロパティを「580, 480」にそれぞれ設定変更する。 | |

　さらに、「MainWindow.cs」ファイル内のコードを次のように修正する。

* 修正前：Application.Run(new **Form1**());
* 修正後：Application.Run(new **MainWindow**());

　以上で、MDIアプリケーションのメイン・ウィンドウの原型は完成だ。次に、このメイン・ウィンドウにメニュー・バーとステータス・バーを追加する。

**■メニュー・バーとステータス・バーの追加**

　メニュー・バーやステータス・バーは、［ツールボックス］ウィンドウにそれぞれ、MainMenuコントロール、StatusBarコントロールとして登録されているので、これらのコントロールをメイン・ウィンドウのフォーム上にドラッグ＆ドロップすれば追加できる。なお、追加したこれらのコントロールの名前は、IDEが自動的に決定した「mainMenu1」「statusBar1」のままにする。

　次の画面は実際にメニュー・バーとステータス・バーをフォーム上に追加したところだ。

|  |
| --- |
|  |
| **メニュー・バーとステータス・バーのメイン・ウィンドウへの追加** |
| ［ツールボックス］ウィンドウにあるMainMenuコントロールとStatusBarコントロールをメイン・ウィンドウのフォーム上にドラッグ＆ドロップすれば、フォームにメイン・メニューとステータス・バーを追加できる。 |
| |  |  |  | | --- | --- | --- | |  |  | MainMenuコントロールをフォーム上にドラッグ＆ドロップする。これにより、フォームの上部と、フォーム・デザイナ下部のコンポーネント表示領域にMainMenuコントロールのオブジェクトが表示されるようになる。本稿では、このコントロールの名前（つまり［(Name)］プロパティ）はIDEが自動生成した「mainMenu1」のままにする。 | |  |  | StatusBarコントロールをフォーム上にドラッグ＆ドロップすると、StatusBarコントロール・オブジェクトがフォーム下部にドッキングされた状態（つまり、コントロールのDocプロパティがBottomに設定された状態）で貼り付けられる。本稿では、このコントロールの名前はIDEが自動生成した「statusBar1」のままにする。さらに、StatusBarコントロール・オブジェクトの［Text］プロパティを「ready」と設定変更する。 | |  |  | メニュー・バー上のメニュー項目でテキストを入力すると、その項目のテキスト（＝［Text］プロパティ）に入力される。メニュー項目の詳細は後述するが、すべてのメニュー項目でこの作業を行う。 | |  |  | テキスト入力を行ったすべてのメニュー項目は、その名前（＝［(Name)］プロパティ）も同時に設定する。この例では、「menuItemWindow」という名前を設定している。 | |

　そして、メニュー・バーのメニュー項目としては、次の表のものを入力した。

|  |  |  |  |
| --- | --- | --- | --- |
| **親項目** | | **子項目** | |
| **名前** | **テキスト** | **名前** | **テキスト** |
| menuItemFile | ファイル(&F) | menuItemStartMenu | スタート メニュー(&M) |
|  | - |
| menuItemClose | 終了(&X) |
| menuItemWindow | ウィンドウ(&W) | menuItemCascade | 重ねて表示(&C) |
| menuItemTileHorizontal | 上下に並べて表示 |
| menuItemTileVertical | 左右に並べて表示(&V) |
|  | - |
| menuItemAllClose | すべて閉じる(&A) |
| menuItemHelp | ヘルプ(&H) | menuItemAboutBox | バージョン情報(&A)... |
|  | | | |
| **メニュー・バーに追加したメニュー項目** | | | |
| メニュー・バーに追加されるメニュー項目の実体は、System.Windows.Forms名前空間のMenuItemオブジェクトである。表中の「名前」は、このオブジェクトの［(Name)］プロパティのことで、テキストは［Text］プロパティを意味する。 | | | |

　これらのほとんどのメニュー項目の内容は、Windowsアプリケーションでは一般的なものばかりなのであらためて説明する必要はないだろう。「スタート メニュー」というメニュー項目は通常は見慣れないものだが、これは業務処理を実行するメニュー画面を表示するための項目で、次回で詳しく解説する。

　上の画面や表のテキストにある、これらのメニュー項目で注意してほしいのは、アンダーラインが記述された英数字である。例えば「ウィンドウ(W)」というテキストの中には「W」という英数字があるが、これはアクセス・キーと呼ばれる機能を設定したものだ。英数字に対してこのアクセス・キーを割り当てるには、「&W」のように、英数字の前にアンパサンド（&）を追加するだけである。アクセス・キーは、大文字のアルファベット、もしくは数字で表記し、その文字を半角の丸カッコで囲むのが慣例となっている。例えば、「W」というアクセス・キーは、「(&W)」と記述する。アクセス・キーの詳細は、.NET TIPS「[Windowsアプリケーションでアクセス・キーを割り当てるには？](https://www.atmarkit.co.jp/fdotnet/dotnettips/206winacckey/winacckey.html)」を参照してほしい。

　メニュー項目を分離するライン（以降、分離ライン）は、メニュー項目として「-」（半角のハイフン）を入力すれば設定できる。実際に上記のメニュー項目では、「左右に並べて表示」というメニュー項目と「すべて閉じる」というメニュー項目の間に分離ラインを設定している。

　上記のメニュー項目の（表の）「バージョン情報(&A)...」というテキストにある「...」は、このメニューを実行すると、何らかのダイアログなどが表示されることを意味する。これもWindowsアプリケーションの慣例である。ちなみに、「バージョン情報」というメニュー項目に限っては「...」が省略されることも多い。

　最後にメニュー・バーをフォームに結び付けるために、フォームの［Menu］プロパティに「mainMenu1」を設定する。これにより、実際にフォームにメニュー・バーが組み込まれる（ちなみにステータス・バーは、何の設定も行うことなく、これまでの実装内容だけで組み込まれる）。また、［ウィンドウ］メニュー項目（＝menuItemWindowメニュー項目）の［MdiList］プロパティを「True」に設定変更する。これにより、メニュー・バーの［ウィンドウ］の下に、表示中の子ウィンドウがリスト表示されるようになる。

　以上で、メニュー・バーとステータス・バーの追加は完了である。

**■子ウィンドウの追加**

　次にメニュー項目にある処理を実装したいが、MDIのメイン・ウィンドウだけで、子ウィンドウが1つもないとそれらの実装状況を確認するのに不都合だ（例えば、子ウィンドウを並べ替える処理を実装しても、子ウィンドウが存在しなければ並べ替えることは不可能である）。そこで一時的な仮のウィンドウとして、次の3つのウィンドウ（のソース・ファイル）を追加してみよう。

* 「StartMenuWindow.cs」（次回使用する）
* 「Form1.cs」（一時的な仮ウィンドウ）
* 「Form2.cs」（一時的な仮ウィンドウ）

　これらのウィンドウを追加する手順は、次の画面のとおりだ。

|  |
| --- |
|  |
| **［新しい項目の追加］ダイアログによるWindowsフォームの追加** |
| ［新しい項目の追加］ダイアログを使って、新たなWindowsフォーム（この例では「StartMenuWindow.cs」）を追加する。 |
| |  |  |  | | --- | --- | --- | |  |  | VS.NET IDEの［ソリューション エクスプローラ］にある「Kokyaku」プロジェクトを右クリックしてコンテキスト・メニューを表示し、そのメニューから［追加］－［新しい項目の追加］を選択すると、［新しい項目の追加］ダイアログが表示される。このダイアログから新たなWindowsフォームを追加できる（本稿の例では、前述の「StartMenuWindow.cs」「Form1.cs」「Form2.cs」の3つを追加する）。 | |  |  | ［テンプレート］から［Windows フォーム］を選択する。 | |  |  | ［ファイル名］欄に、Windowフォームのファイル名（この例では「StartMenuWindow.cs」）を記入する。このファイル名から拡張子（この例では「.cs」）を除いたものが、クラス名（この例では「StartMenuWindow」）として使われる。 | |  |  | ［開く］ボタンをクリックすると、新しいWindowsフォームが新規作成される。 | |

　子ウィンドウが用意できたら、メニュー項目の処理を実装していこう。

**■メニュー項目の処理の実装**

　メニュー項目の処理を実装するには、それぞれのメニュー項目のClickイベント・ハンドラを追加する必要がある。これには、先ほど作成したメニュー項目をそれぞれダブルクリックしていけばよい（ただし親メニュー項目である［ファイル］、［ウィンドウ］、［ヘルプ］は子ニュー項目をまとめるだけのものなので、これらに対するClickイベント・ハンドラは追加しなくてよい）。さらに、フォームをダブルクリックして、Loadイベント・ハンドラも同時に追加する。すべてのイベント・ハンドラの追加が終われば、各ハンドラに次のコードを実装しよう。

|  |  |
| --- | --- |
| |  | | --- | | private void menuItemStartMenu\_Click(object sender, System.EventArgs e) {   StartMenuWindow startMenu = new StartMenuWindow();   startMenu.MdiParent = this;   startMenu.WindowState = FormWindowState.Maximized;   startMenu.Show();   startMenu.Update(); }  private void menuItemClose\_Click(object sender, System.EventArgs e) {   this.Close(); }  private void menuItemCascade\_Click(object sender, System.EventArgs e) {   this.LayoutMdi(MdiLayout.Cascade); }  private void menuItemTileHorizontal\_Click(object sender, System.EventArgs e) {   this.LayoutMdi(MdiLayout.TileHorizontal); }  private void menuItemTileVertical\_Click(object sender, System.EventArgs e) {   this.LayoutMdi(MdiLayout.TileVertical); }  private void menuItemAllClose\_Click(object sender, System.EventArgs e) {   Form [] children = this.MdiChildren;   foreach (Form child in children)   {     child.Close();     child.Dispose();   } }  private void menuItemAboutBox\_Click(object sender, System.EventArgs e) {   MessageBox.Show("顧客.NET\n  ver. 0.1"); }  private void MainWindow\_Load(object sender, System.EventArgs e) {   StartMenuWindow startMenu = new StartMenuWindow();   startMenu.MdiParent = this;   startMenu.WindowState = FormWindowState.Maximized;   startMenu.Show();   startMenu.Update();    Form1 winform1 = new Form1();   winform1.MdiParent = this;   winform1.WindowState = FormWindowState.Maximized;   winform1.Show();   winform1.Update();    Form2 winform2 = new Form2();   winform2.MdiParent = this;   winform2.WindowState = FormWindowState.Maximized;   winform2.Show();   winform2.Update(); } | |
| **メニュー項目の処理を行うサンプル・コード（C#）** |
| VB.NETのサンプル・コード（[winexp01\_01.vb](https://www.atmarkit.co.jp/fdotnet/winexp/winexp01/winexp01_01.vb)） |

　これらのコードは非常に単純なので詳しく説明する必要はないだろう。よってここでは、ごく簡単に説明するにとどめる。

　menuItemStartMenuコントロール（メニュー項目の［スタート メニュー］）のClickイベント・ハンドラと、MainWindow（メイン・ウィンドウ）のLoadイベント・ハンドラでは、それぞれの子ウィンドウのインスタンスを生成して表示する処理を実装している。ここで注意が必要なのは、そこで生成した子ウィンドウ・オブジェクトのMdiParentプロパティに、メイン・ウィンドウのオブジェクト（this）を設定しているところだ。これにより、そこで生成されたウィンドウがMDIアプリケーションの子ウィンドウとして機能する。

　menuItemCascadeコントロール（［重ねて表示］メニュー項目）、menuItemTileHorizontalコントロール（［上下に並べて表示］メニュー項目）、menuItemTileVerticalコントロール（［左右に並べて表示］メニュー項目）のClickイベント・ハンドラでは、フォームのLayoutMdiメソッドによりウィンドウの整列を行っている。LayoutMdiメソッドについては、[MSDNのドキュメント](http://www.microsoft.com/japan/msdn/library/ja/cpref/html/frlrfsystemwindowsformsformclasslayoutmditopic.asp" \t "_blank)を当たってほしい。

　menuItemAllCloseコントロール（メニュー項目の［すべて閉じる］）のClickイベント・ハンドラでは、すべてのMDIアプリケーションの子ウィンドウを1つずつ取り出して、それらを順に閉じていく処理を行っている。

　以上でメニュー項目の処理を実装できた。ここまでで実装してきたアプリケーションを実際に実行してみよう。

|  |
| --- |
|  |
| **実装したMDIアプリケーションの実行画面** |

　MDIアプリケーションを実行すると、メニュー項目の処理はすべて正常に実行できることが確認できる。しかし、1点だけ、開発者の意図どおりに動いていない個所がある。それは、メニュー・バーの［スタート メニュー］を選択すると、StartMenuWindowウィンドウがいくつも表示できてしまうことだ。次に、この問題を修正するために、StartMenuWindowウィンドウに対して1つのウィンドウしか表示しない仕組みを追加することにしよう。

**■シングルトンの実装**

　オブジェクトが1つしか生成できないことを保証する設計の仕組みは、デザイン・パターン（＝よく使う汎用的な設計パターンをまとめたもの）の世界では、「シングルトン・パターン」と呼ばれる。このシングルトン・パターンをStartMenuWindowフォームに適用すれば先ほどの問題を解決することができる。具体的には、「StartMenuWindow.cs」に次のコードを加筆・修正する（追加・変更するのは太字の部分のみ）。

|  |  |
| --- | --- |
| |  | | --- | | **// クラス外部からのnew演算子の使用を禁止** **//public StartMenuWindow()** **private StartMenuWindow()** {   InitializeComponent(); }  protected override void Dispose( bool disposing ) {   if( disposing )   {     if(components != null)     {       components.Dispose();     }   }   base.Dispose( disposing );    **// ウィンドウ破棄の際にインスタンスを初期化する**   **instance = null;** }  **#region Windows フォームをシングルトン対応にするコード** **// インスタンスを格納するフィールド変数** **private static StartMenuWindow instance = null;**  **// インスタンスを取得するためのメソッド** **public static StartMenuWindow GetInstance()** **{**   **if (instance == null)**   **{**     **instance = new StartMenuWindow();**   **}   return instance;** **} #endregion** | |
| **Windowsフォームにシングルトンを適用するサンプル・コード（C#）** |
| VB.NETのサンプル・コード（[winexp01\_02.vb](https://www.atmarkit.co.jp/fdotnet/winexp/winexp01/winexp01_02.vb)） |

　このコードでは、StartMenuWindowクラスのインスタンスを生成する手段を静的メソッドの「GetInstanceメソッド」のみに限定し、さらにそのインスタンスを静的フィールド変数の「instanceフィールド変数」に1つだけ保持・管理するような仕組みとなっている。これにより、クラス構造のレベルで、StartMenuWindowオブジェクトが確実に1つしか存在しないことを保証する。

　たとえもし、クラス外部でnew演算子によりStartMenuWindowオブジェクトを生成しようとするコードを記述しても、StartMenuWindowクラスのコンストラクタのアクセス修飾子が「private」となっているために、クラス外部からStartMenuWindowクラスのコンストラクタにアクセスできず、コンパイル・エラーとなる。

　よって、new演算子を使ってStartMenuWindowオブジェクトを生成していた先ほどのコードは、StartMenuWindow.GetInstanceメソッドを使ったコードに置き換える必要がある。具体的には次のように修正する。

* 修正前：  
    StartMenuWindow startMenu = **new StartMenuWindow();**
* 修正後：  
    StartMenuWindow startMenu = **StartMenuWindow.GetInstance();**

　本稿でこの修正が必要な個所は次の2カ所である。

* menuItemStartMenuコントロール（［スタート メニュー］メニュー項目）のClickイベント・ハンドラ
* MainWindow（メイン・ウィンドウ）のLoadイベント・ハンドラ

　これらの修正を行って再度プログラムを実行すると、StartMenuWindowウィンドウが常に1つしか起動しないのを確認できる。もちろん本稿のようにわざわざシングルトンを使わなくても、同様の処理は実現可能である。例えば、子ウィンドウのインスタンス（＝この例では「StartMenuWindowウィンドウ」）が作成されたかどうかのフラグを、メイン・ウィンドウのクラス内に作成して、そのフラグを使って（メイン・ウィンドウ側で）子ウィンドウのインスタンスが1つしか生成されないように管理すればよい。しかし、それではメイン・ウィンドウと子ウィンドウの依存性が増してしまう。2つのクラスの依存性が増すと、一方のクラスの変更が、もう一方のクラスに（知らないうちに）影響を与える可能性が高くなる。よって、本稿のようにシングルトン・パターンを使ってクラス内で自身のインスタンス管理を完結させた方が、（ほかのクラスからの影響に対して）より頑健なクラスになるといえるだろう。

■

　以上で今回は終わりだ。ここまでの実装でMDI型Windowsアプリケーションの基礎部分が作成できたことになる。次回以降の解説で、さらにこのMDIアプリケーションを拡張していく予定だ。

　次回は、業務アプリケーションでは欠かせない「使いやすさを追求したスタート・メニュー画面の構築」について解説する。スタート・メニュー画面とは、業務処理画面に簡単にアクセスするためのメニュー画面である。次回もお楽しみに。[End of Article](https://www.atmarkit.co.jp/fdotnet/index.html)