【セッションNo. 4】

Delphi/400技術セッション

開発者が知りたい実践プログラミングテクニック！

株式会社ミガロ.
RAD事業部　営業・営業推進課
尾崎　　浩司

【アジェンダ】

- 『継承』について

- 『継承』を使用した開発手法
  1. フォームの『継承』
  2. コンポーネントの『継承』

  （補足）TObjectを『継承』した業務ロジック一元化

- まとめ

『継承』について

■ オブジェクト指向とは？

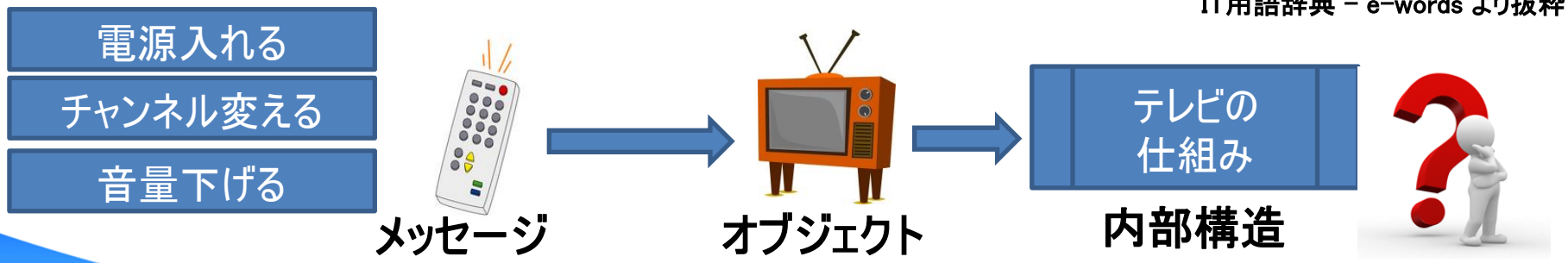
オブジェクト指向とは、ソフトウェアの設計や開発において、操作手順よりも操作対象に重点を置く考え方。

関連するデータの集合と、それに対する手続き（メソッド）を「オブジェクト」と呼ばれる一つのまとまりとして管理し、その組み合わせによってソフトウェアを構築する。

すでに存在するオブジェクトについては、利用に際してその内部構造や動作原理の詳細を知る必要はなく、外部からメッセージを送れば機能するため、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る。

データやその集合を現実世界の「モノ」になぞらえた考え方であることから、「オブジェクト」指向と呼ばれる。

IT用語辞典 － e-words より抜粋



■ Delphi/400でのオブジェクト

- オブジェクトの代表は、コンポーネント
  - コンポーネントの中身を知らなくても、使い方さえわかれば使用できる
    - プロパティ　…　コンポーネントの外観や挙動に影響を与える
    - イベント　…　コンポーネントで発生した出来事
    - メソッド　…　コンポーネントの中で実行される一連のサブルーチン

色を変える

Colorプロパティ

文字をセットする

Textプロパティ

コピーする

CopyToClipboardメソッド

値が書き換わる

OnChangeイベント

オブジェクト

TEditの仕組み

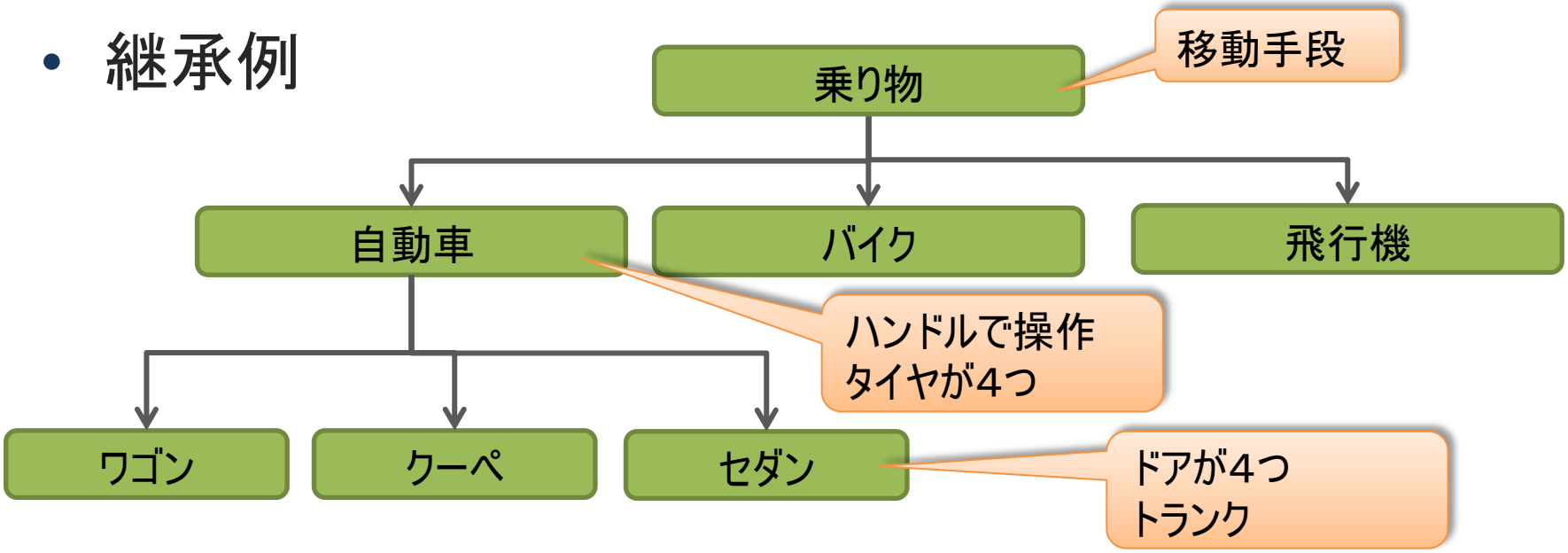
内部構造

■ 『継承』について

- オブジェクト指向　3大要素の一つ

カプセル化	オブジェクト内の詳細仕様や構造を外部から隠蔽すること
継承	存在するクラスを元に、拡張した新しいクラスを定義すること
ポリモーフィズム（多態性）	同名のメソッド等をオブジェクトの種類に応じて使い分けること

- 継承例



『継承』により、上位クラスの機能を引き継いで新たな機能拡張が可能

■ 今回のトピックは、『継承』

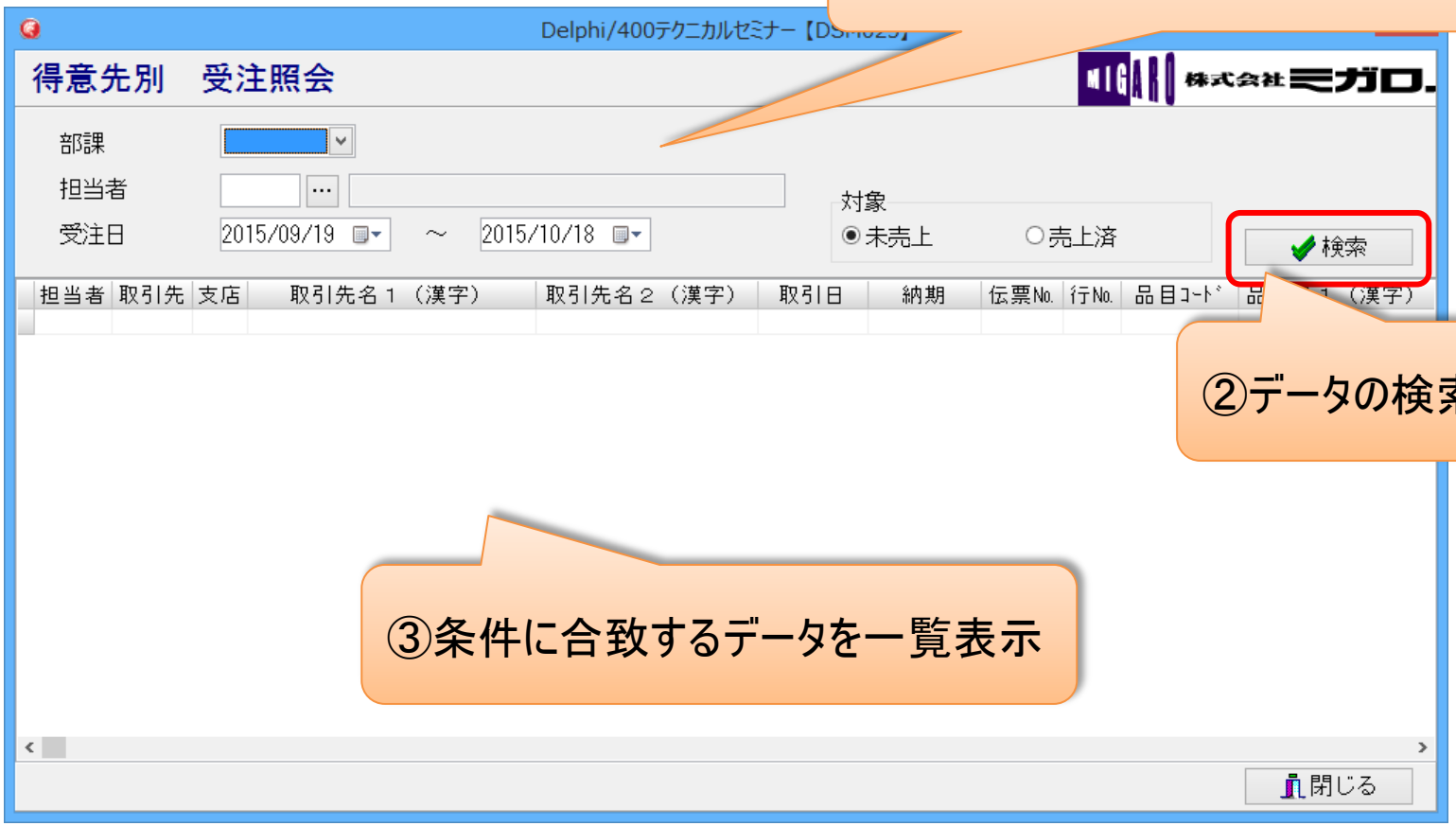
- 『継承』を使用した開発効率化を図る手法を2つ紹介
  - フォームの『継承』
    - フォームを継承することにより、画面のタイプごとに開発手法を統一化
  - コンポーネントの『継承』
    - コンポーネントを継承することにより、汎用的な処理を部品化

『継承』を使用した開発手法

1. フォームの『継承』

■ 一般的な業務アプリケーション

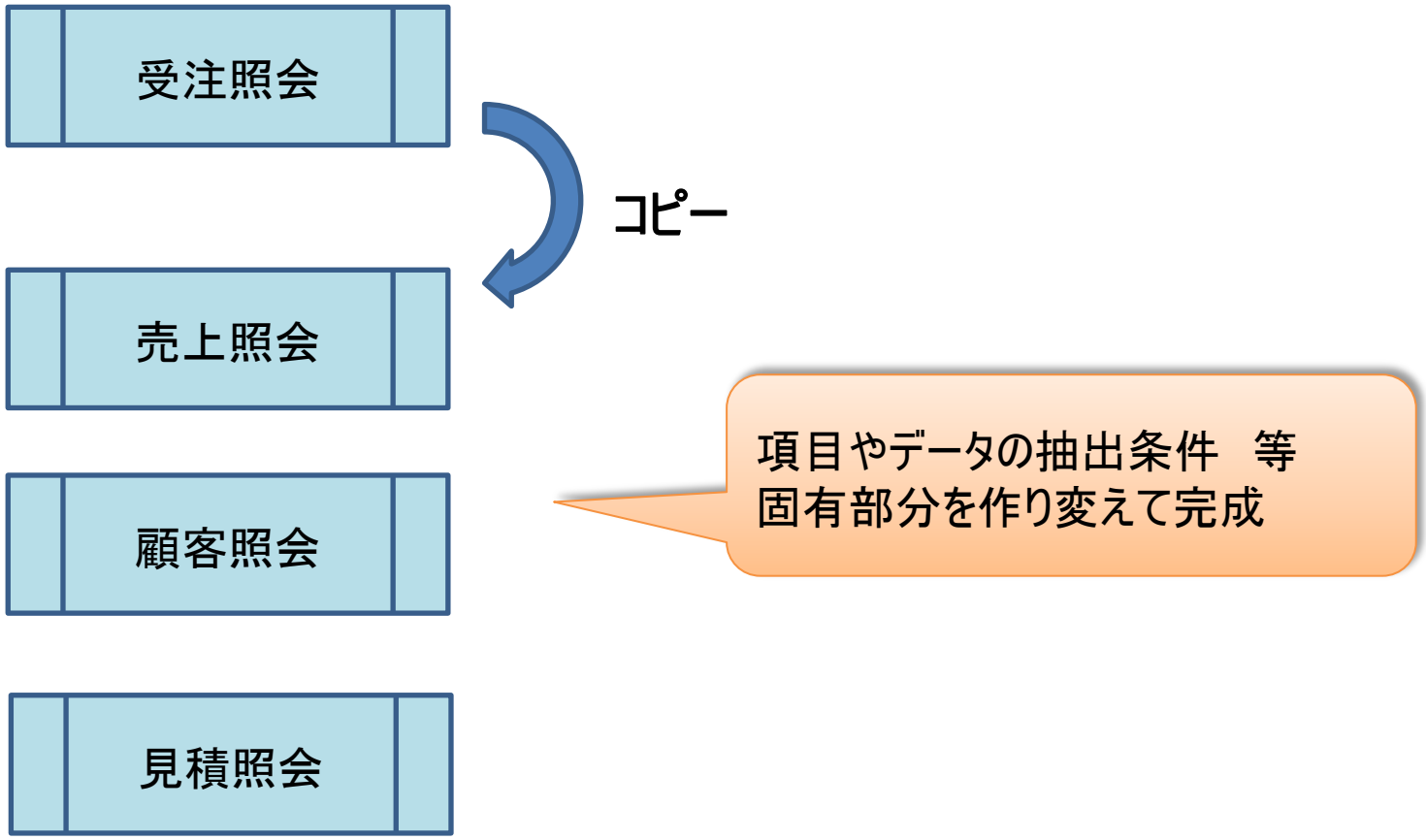
- データ照会画面の場合



画面パターンによって、画面構成や処理手順が似ていることが多い！

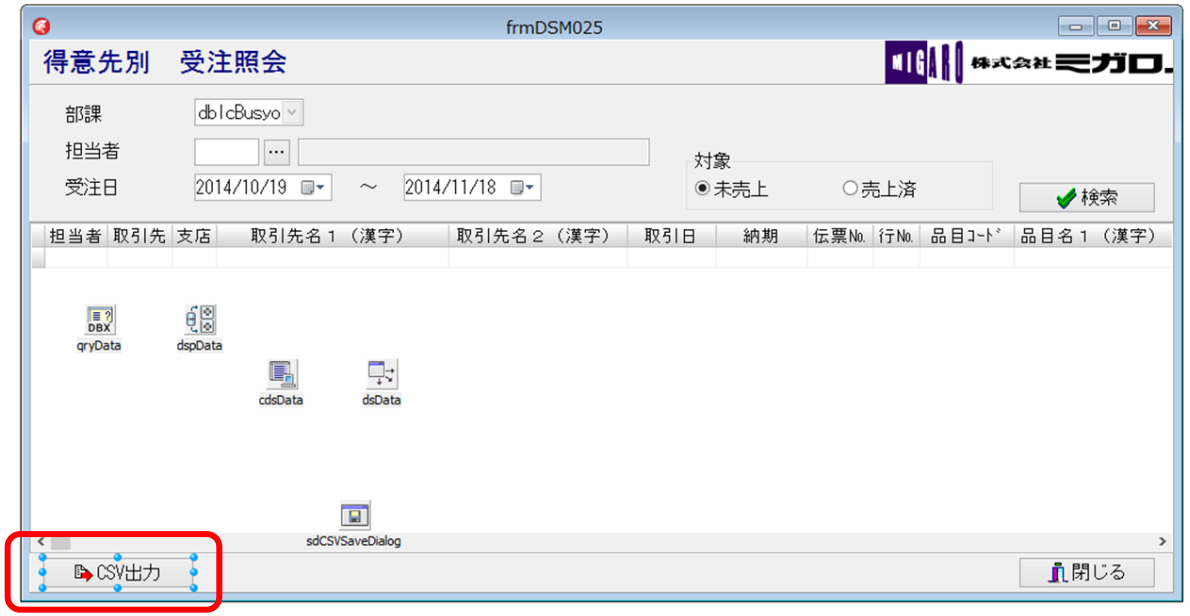
■ 類似機能の画面を構築する場合

- フォームやソースをコピー＆ペーストしても良いが…



■ 機能追加時の対応

- もし、全部の照会画面にCSV出力機能の追加が必要になったら…



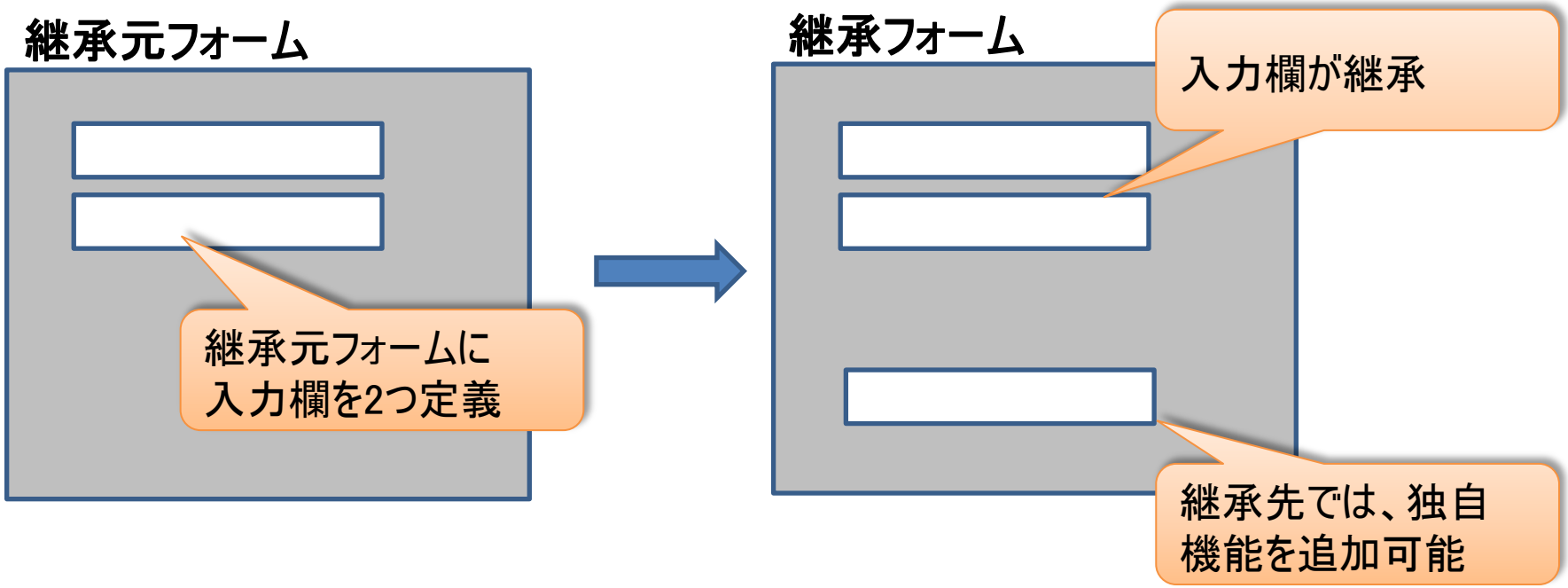
受注照会　売上照会　顧客照会　…

同じ処理をすべてのプログラムに
追加していかないといけない

追加機能

■ 継承フォームとは

- 元のフォーム機能をそのまま持つフォーム。継承元で定義した機能は、継承先でそのまま利用可能。



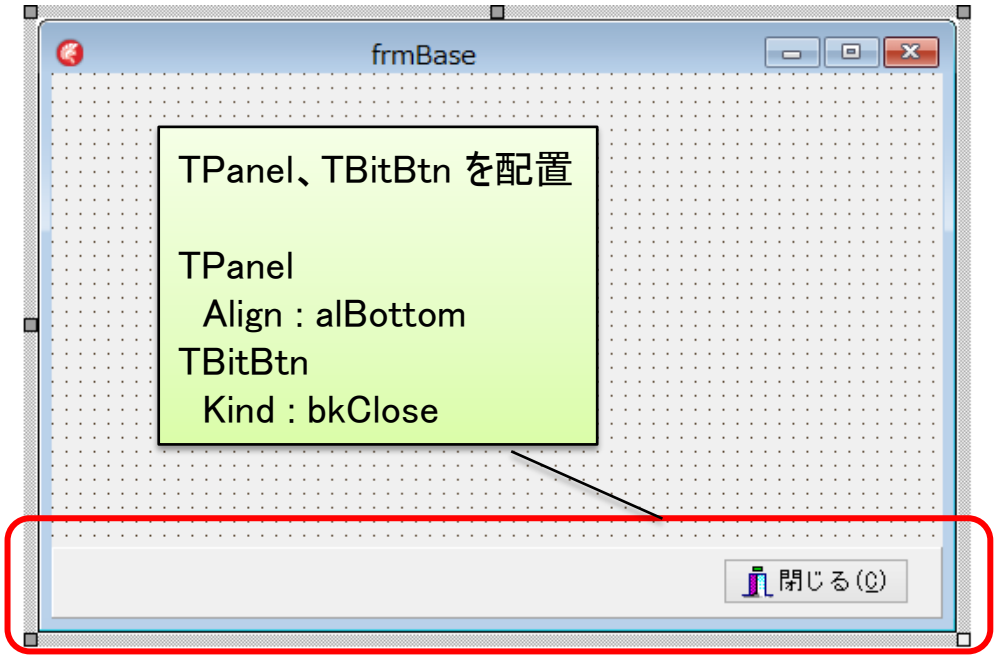
■ 継承フォームの作成方法

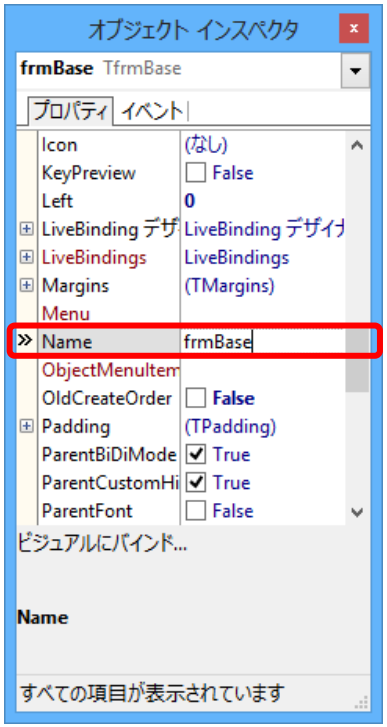
- プロジェクトで、継承元フォームを作成

  例：

  FormのNameプロパティ　→　frmBase

  [ファイル]→[名前を付けて保存]　→　“BaseFrm.pas”



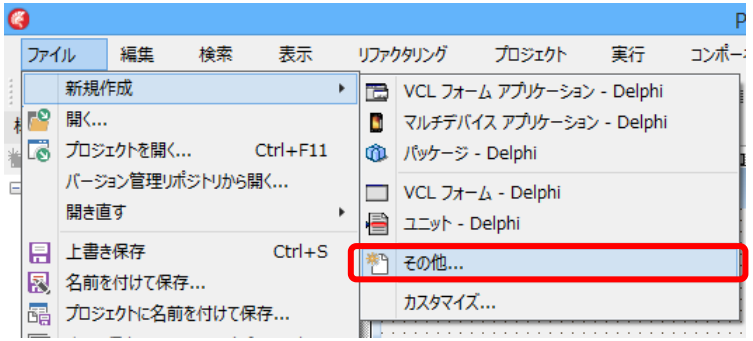


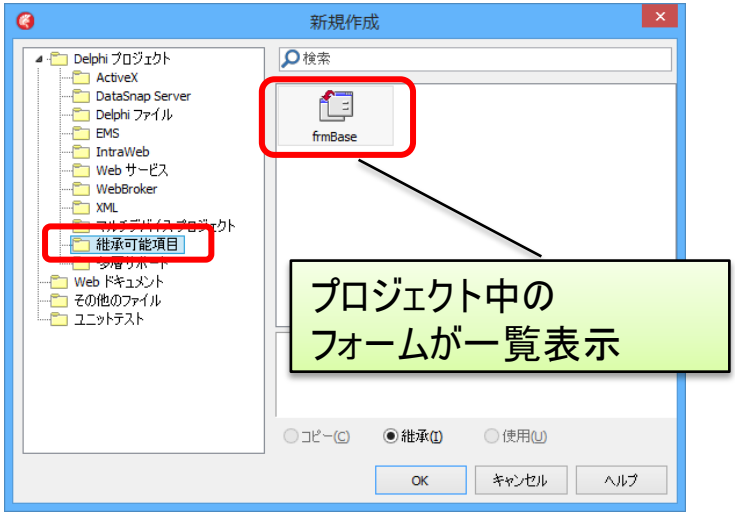
■ 継承フォームの作成方法

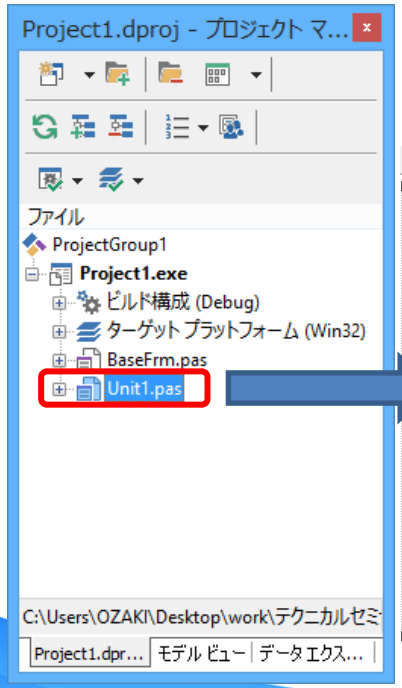
- 継承フォームの新規作成

  [ファイル]→[新規作成]→[その他] を選択

  新規作成ダイアログより、
  [継承可能項目]→[frmBase]を選択



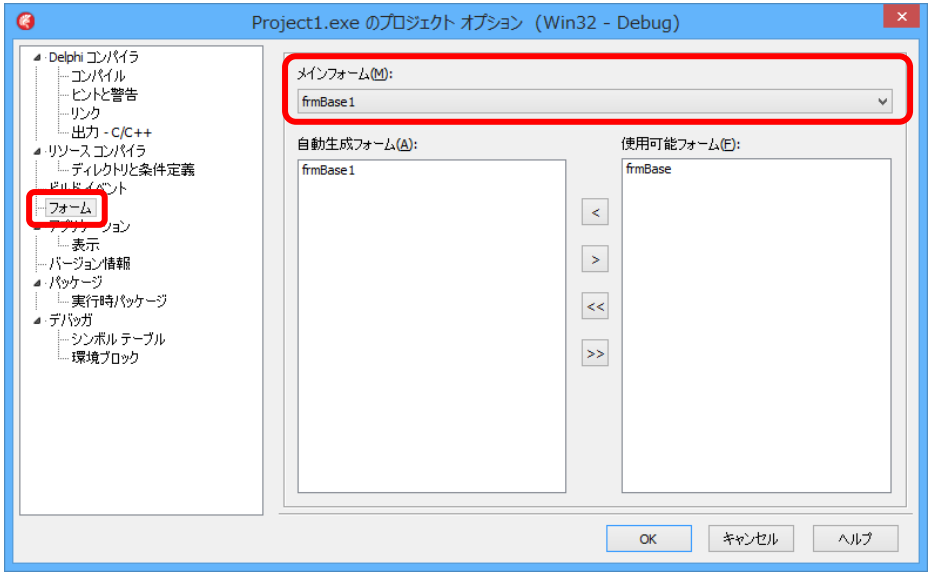




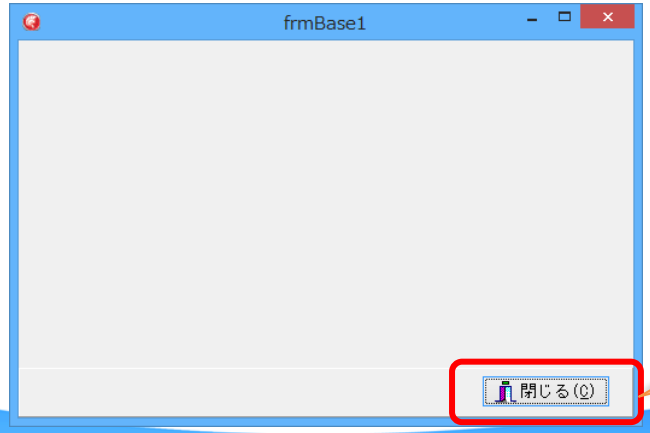
■ 継承フォームの実行

- メインフォームをfrmBase1に設定して実行

[プロジェクト]→[オプション]
フォームより
メインフォーム：frmBase1
frmBaseは使用可能フォーム



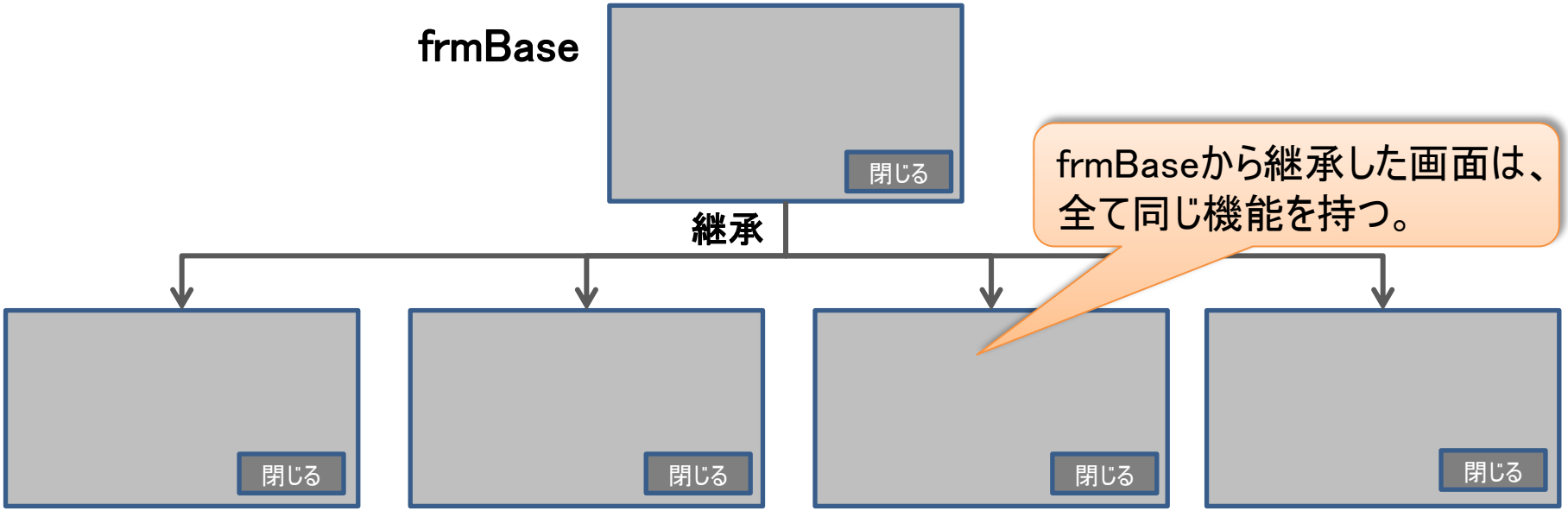
【実行】
「閉じる」ボタンを押下すると、アプリケーションが終了



frmBaseで定義した「閉じる」機能が、そのまま継承先のfrmBase1でも有効

■ 継承フォームの効果

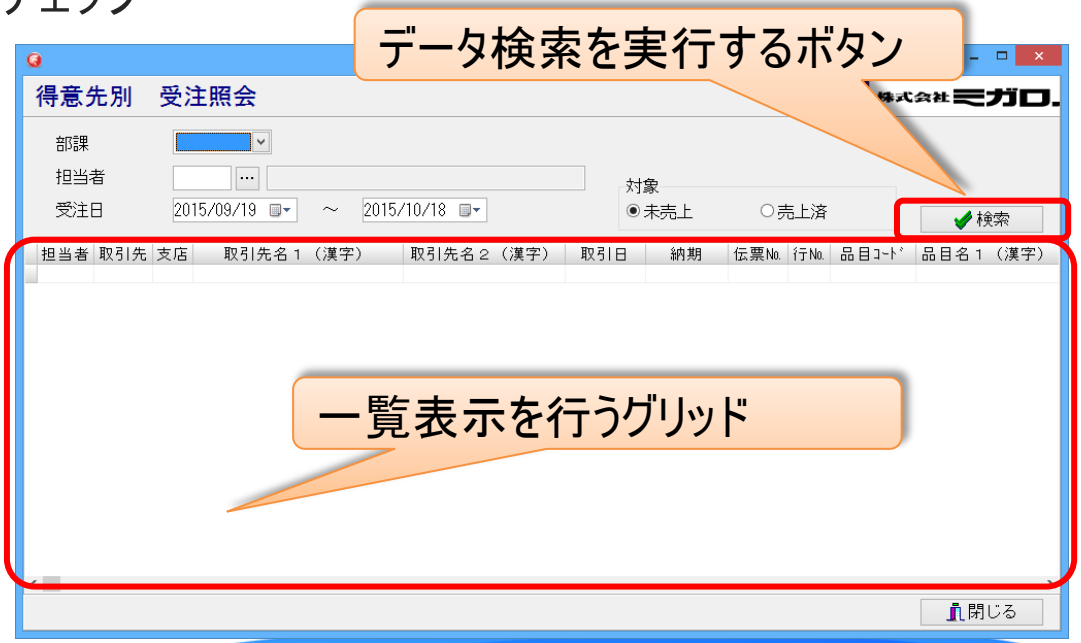
- 継承元フォームに共通機能を定義しておくと、継承先フォームは自動で利用可能になる



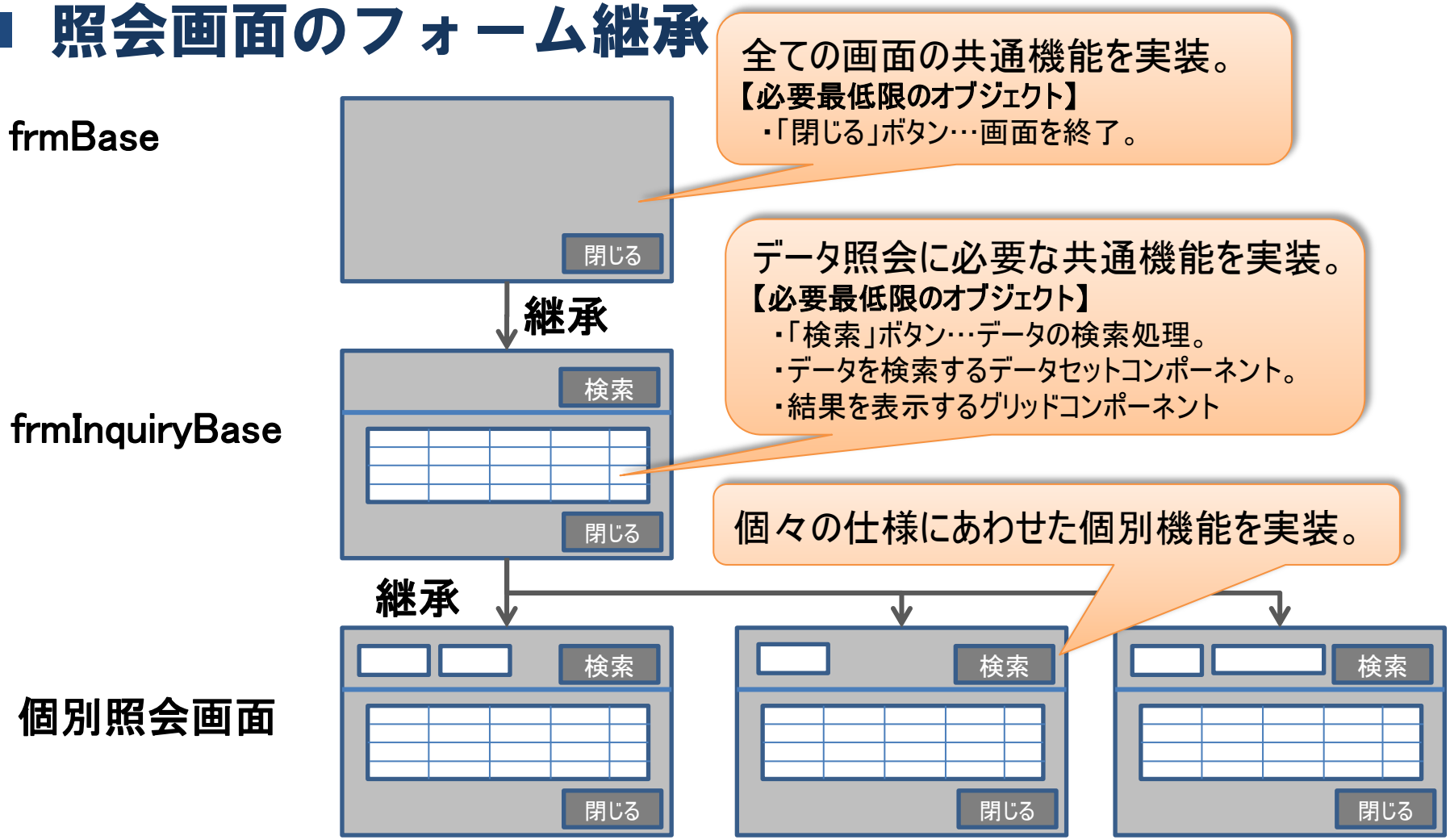
共通機能の機能変更がある場合、frmBaseのみ修正すれば良い

■ フォーム継承を利用した照会画面の検討

- 共通となる機能
  - 画面
    - データ検索を実行する為の「検索」ボタン
    - 検索結果を一覧表示するグリッド
  - 必要な処理
    - 「検索」ボタン押下
      - 入力条件の妥当性チェック
      - SQLの組み立て
      - SQL実行
      - 結果の表示



■ 照会画面のフォーム継承

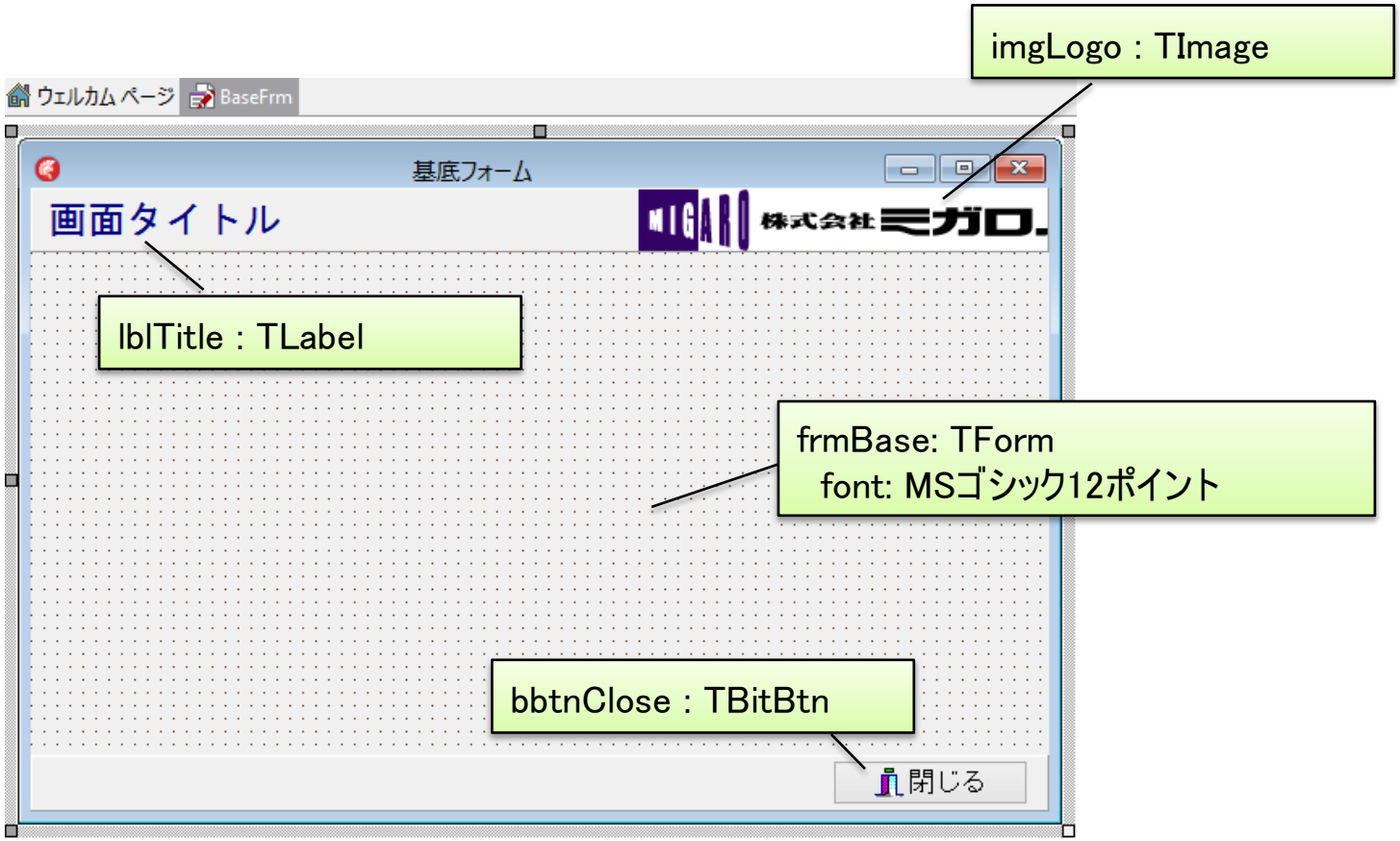


※ 継承元に貼り付けたオブジェクトは、継承先では削除できないので、**どの機能が継承元で必要かを検討することがポイント**

継承元には、必要最低限のオブジェクトを配置するように構成！

frmBaseのレイアウト

- フォーム上に、各画面共通となる部品を配置



■ frmBaseのソースコード

```
implementation

uses ShellAPI;

{$R *.dfm}

const
  cURL = 'http://www.migaro.co.jp/';                    // HP URL

procedure TfrmBase.bbtnCloseClick(Sender: TObject);
begin
  //画面を終了する
  Close;
end;

procedure TfrmBase.FormCreate(Sender: TObject);
begin
  //フォームタイトル設定
  Caption := Application.Title + '  【' + Self.Name + '】';

  //最小サイズの決定
  Constraints.MinHeight := Height;
  Constraints.MinWidth := Width;
end;

procedure TfrmBase.imgLogoClick(Sender: TObject);
begin
  //指定したURLを開く
  ShellExecute(0, 'open', PChar(cURL), nil, nil, SW_SHOW);
end;
```

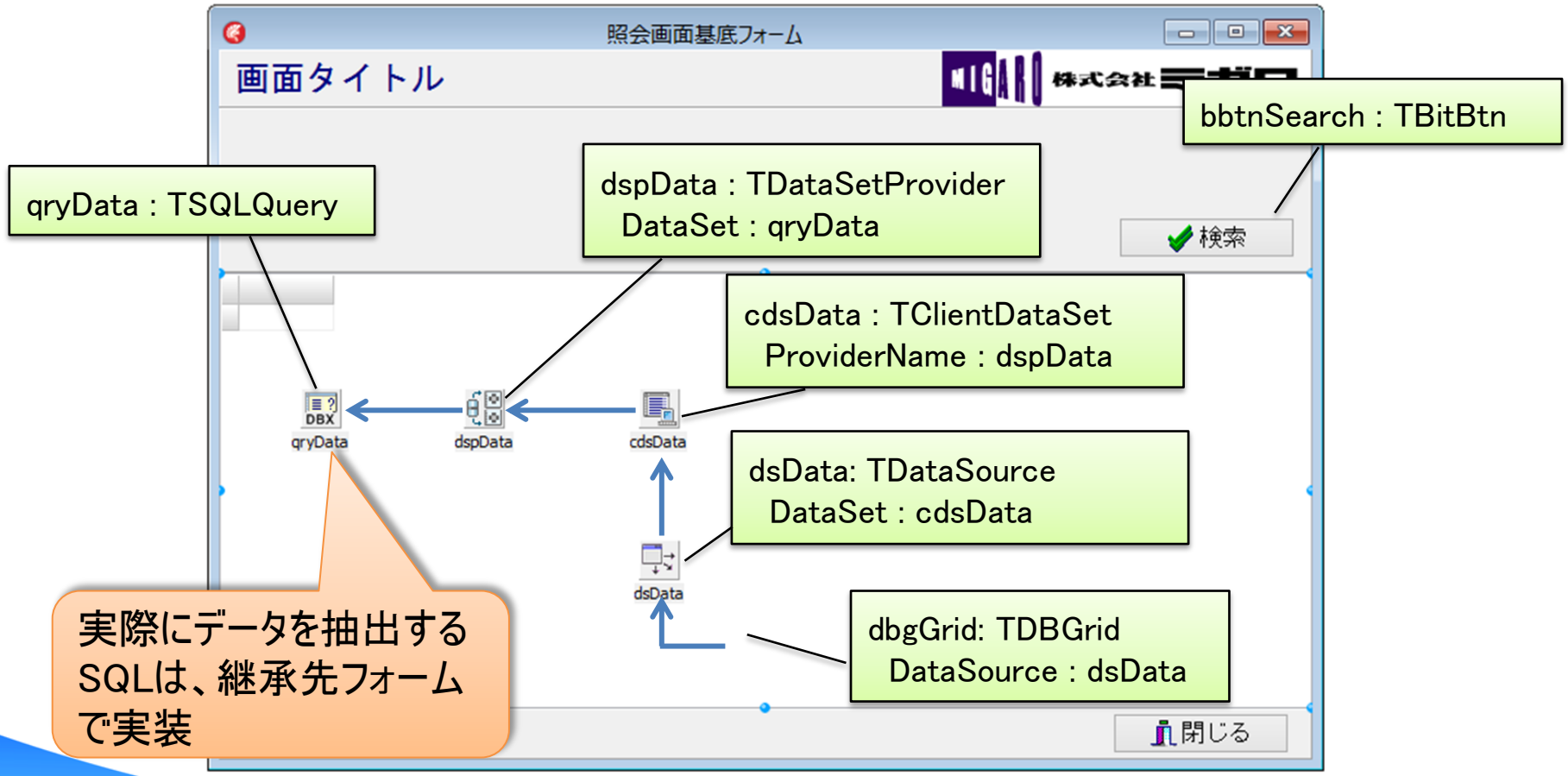
画面を終了する

設計画面の幅・高さを
画面の最小サイズとする。

ロゴクリックにより、ブラウザを
起動し、HPを表示

frmInquiryBaseのレイアウト

- frmBaseを継承し、SQLで抽出する照会画面に必要な部品を配置



frmInquiryBaseのソースコード

```
procedure TfrmInquiryBase.bbtnSearchClick(Sender: TObject);
begin
  inherited;
  //データをクリア
  ClearData;

  //エラーチェック
  if ErrorCheck then Exit;

  //条件セット
  SetQuery;

  //データ抽出
  cdsData.Active := True;

  cdsData.First;
  //対象データが存在しない場合エラー
  if cdsData.Bof and cdsData.Eof then
  begin
    //データをクリア
    ClearData;
    MessageDlg('対象データが存在しません。', mtError, [mbOK], 0);
    Abort;
  end;

  dbgGrid.SetFocus;
end;

procedure TfrmInquiryBase.ClearData;
begin
  //データセットを閉じる
  qryData.Active := False;
  cdsData.Active := False;
end;

procedure TfrmInquiryBase.FormShow(Sender: TObject);
begin
  inherited;
  //データを初期クリアする
  ClearData;
end;
```

ErrorCheck関数で、検索条件に対するエラーチェックを行う。

SetQuery手続きの中で、検索条件にもとづき、TSQLQueryのSQLを作成する。

TSQLQueryと紐づくクライアントデータセットをオープンしデータを抽出する。

データセットをクローズする。

ErrorCheck関数
SetQuery手続き
の実装がない？

frmInquiryBaseの宣言部

- 継承元でサブルーチンの呼出しは行うが、サブルーチンの処理実装自体は、継承先フォームで作成

```
type
  TfrmInquiryBase = class(TfrmBase)
    pnlKey: TPanel;
    bbtnSearch: TBitBtn;
    dbgGrid: TDBGrid;
    qryData: TSQLQuery;
    dspData: TDataSetProvider;
    cdsData: TClientDataSet;
    dsData: TDataSource;
    procedure FormShow(Sender: TObject);
    procedure bbtnSearchClick(Sender: TObject);
  private
    { Private 宣言 }
  protected
    procedure ClearData;
    function ErrorCheck: Boolean; virtual; abstract;
    procedure SetQuery; virtual; abstract;
  public
    { Public 宣言 }
  end;
```

```
// エラーチェック
// 条件セット
```

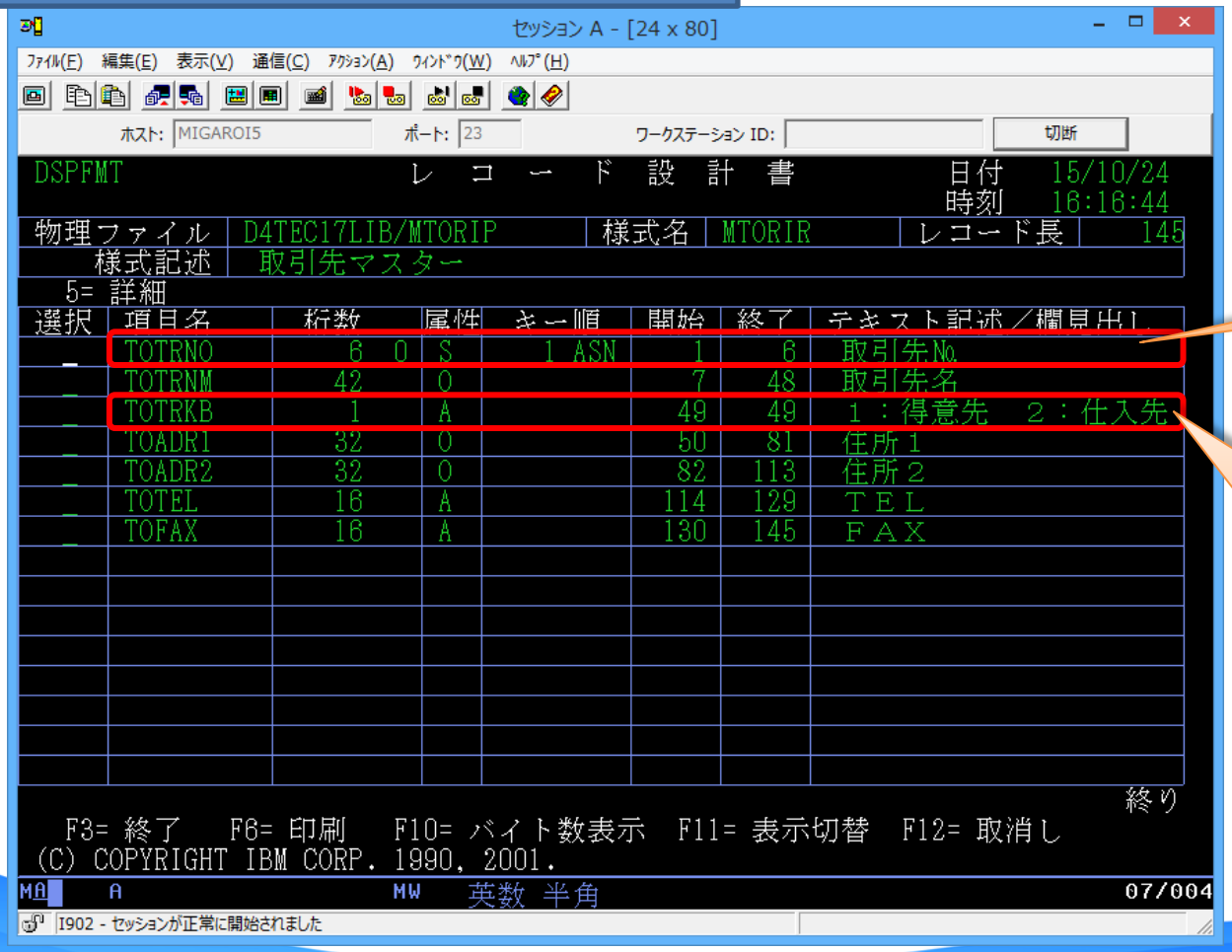
virtual; abstract ： **抽象メソッド**
サブルーチンの宣言は行うが、実際の実装は下位クラス（継承先フォーム）で行う。

継承先での実装部の処理を宣言する場合、abstractを使用

■ 照会画面の作成

- 取引先マスターー覧照会をfrmInquiryを継承して作成

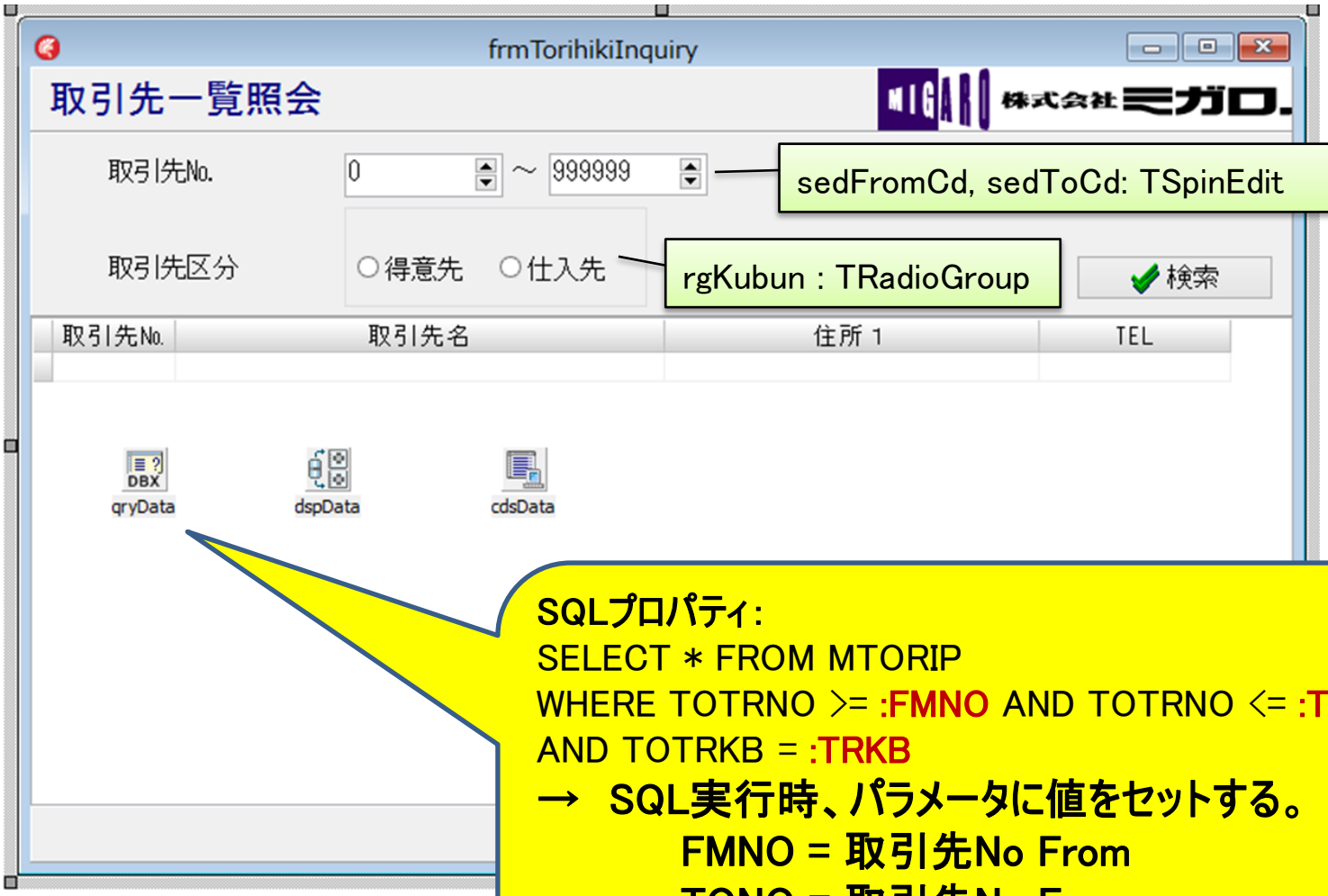
MTORIP(取引先マスター)



TOTRNO
取引先No ： キー項目

TOTRKB
取引先区分
1:得意先 / 2:仕入先

■ 取引先一覧照会のレイアウト



SQLプロパティ:
SELECT * FROM MTORIP
WHERE TOTRNO >= :FMNO AND TOTRNO <= :TONO
AND TOTRKB = :TRKB
→ SQL実行時、パラメータに値をセットする。
FMNO = 取引先No From
TONO = 取引先No From
TRKB = 取引先区分

■ 取引先一覧照会のソースコード

- 継承先の宣言部にて継承元サブルーチンをoverride

```
type
  TfrmTorihikiInquiry = class(TfrmInquiryBase)
    Label1: TLabel;
    sedFromCd: TSpinEdit;
    sedToCd: TSpinEdit;
    Label2: TLabel;
    Label3: TLabel;
    rgKubun: TRadioGroup;
    cdsDataTOTRNO: TIntegerField;
    cdsDataTOTRNM: TStringField;
    cdsDataTOTRKB: TStringField;
    cdsDataTOADR1: TStringField;
    cdsDataTOADR2: TStringField;
    cdsDataTOTEL: TStringField;
    cdsDataTOFAX: TStringField;
    procedure sedFromCdChange(Sender: TObject);
    procedure sedToCdChange(Sender: TObject);
    procedure rgKubunClick(Sender: TObject);
  private
    { Private 宣言 }
  public
    { Public 宣言 }
    function ErrorCheck: Boolean; override;
    procedure SetQuery; override;
  end;
```

```
// エラーチェック
// 条件セット
```

override ： **メソッドの再定義**
上位クラスで宣言されたサブルーチンを
下位クラスで再定義

継承元でabstract宣言されたサブルーチンは、継承先で必ず
overrideしたサブルーチンを作成する

■ 取引先一覧照会のソースコード

- 実装ロジックに業務ロジックを追加

```
function TfrmTorihikiInquiry.ErrorCheck: Boolean;
begin
  Result := False;
  //取引先コード　大小チェック
  if sedFromCd.Value > sedToCd.Value then
  begin
    MessageDlg('コードの大小が正しくありません', mtError, [mbOk], 0);
    Result := True;
    Exit;
  end;
  //取引先区分入力チェック
  if rgKubun.ItemIndex < 0 then
  begin
    MessageDlg('取引先区分が指定されてません。', mtError, [mbOK], 0);
    Result := True;
    Exit;
  end;
end;

procedure TfrmTorihikiInquiry.SetQuery;
var
  sKubun: String; // 取引先区分
begin
  sKubun := IntToStr(rgKubun.ItemIndex + 1);  //1:得意先、2:仕入先

  //SQLにパラメータをセット
  with qryData do
  begin
    ParamByName('FMNO').AsInteger := sedFromCd.Value;   // 取引先FROM
    ParamByName('TONO').AsInteger := sedToCd.Value;     // 取引先TO
    ParamByName('TRKB').AsString := sKubun;             // 取引先区分
  end;
end;
```

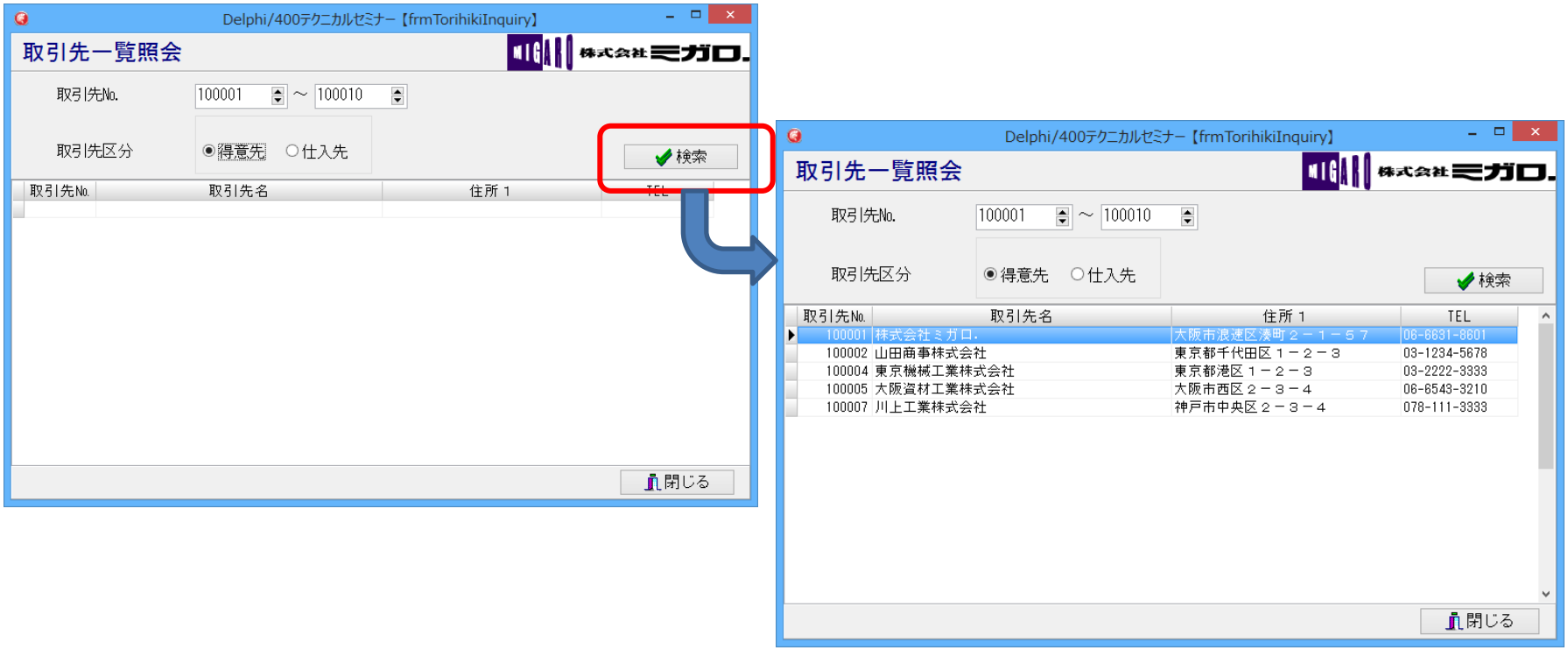
画面上のエラーチェックを
行い、エラーがある場合、
メッセージを表示し、
Result = Trueをセット

**Trueを返すと検索処理が
中断**

パラメータクエリーに
画面上の値をセットして
SQL文を完成させる。

■ アプリケーションの実行

- 画面の動作は、継承元で定義されている為、個別機能のみを継承フォームで定義すれば良い。

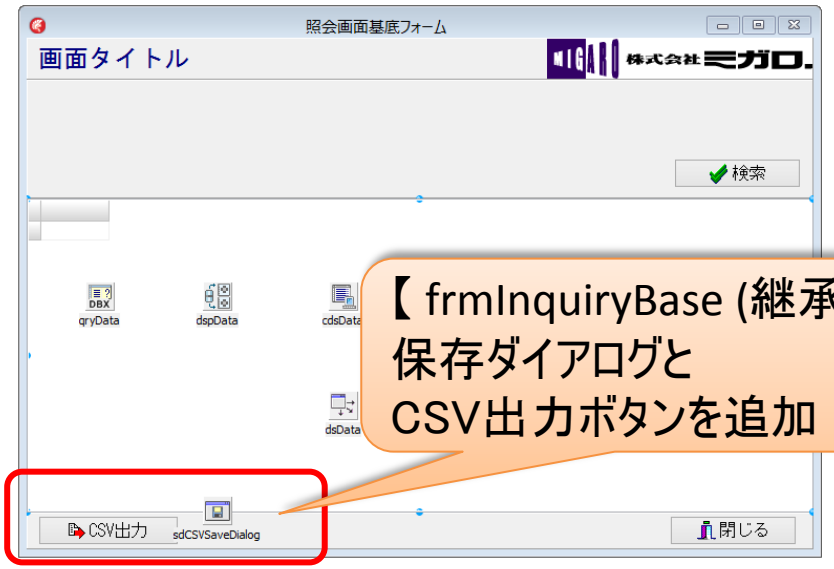


同じ動作をする画面はすべて同じ手法で開発可能

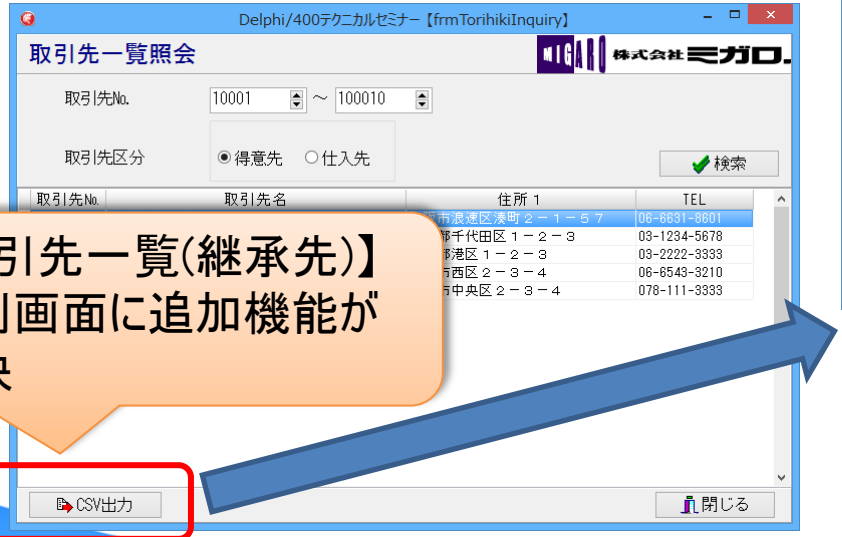
→ 開発効率向上、 動作統一化

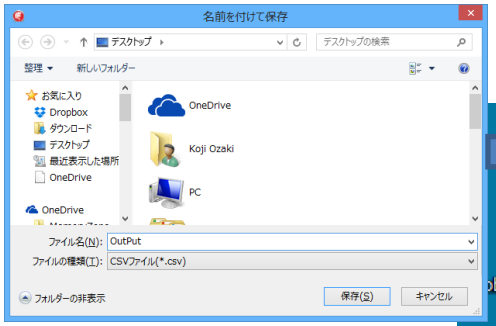
■ 継承元機能拡張例

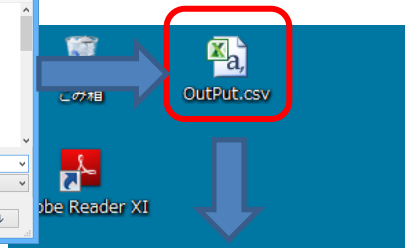
- 「CSV出力」機能の追加
  - ソースはサンプルCDに収録

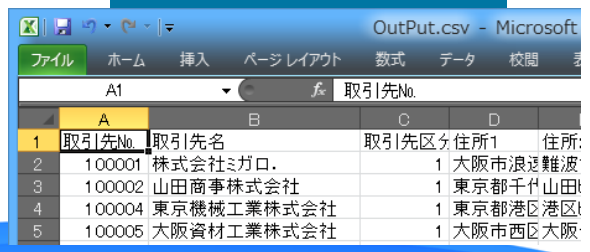


- 実行



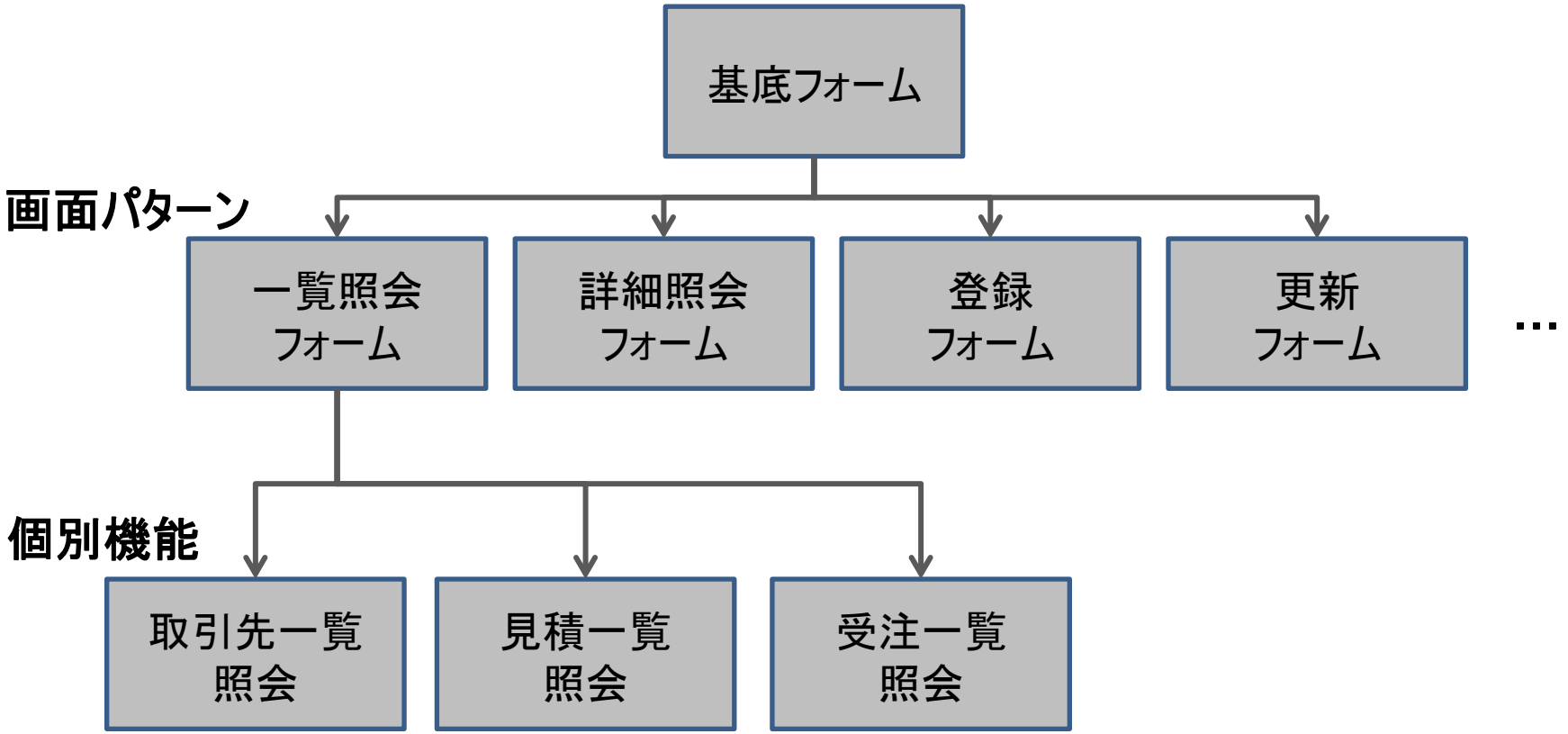






■ フォームの『継承』

- フォーム継承例

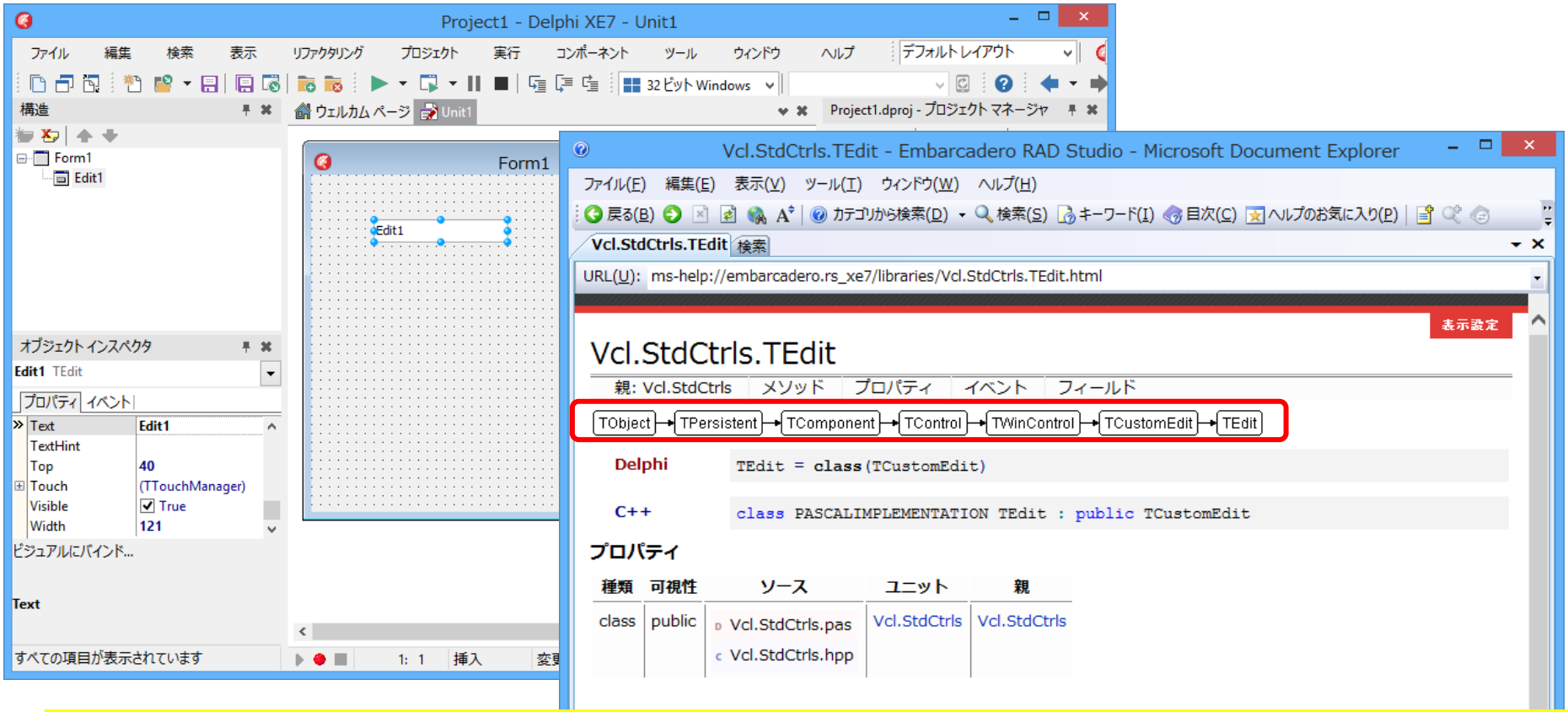


業務アプリを画面パターンに分けることで、仕様の統一化
開発の統一化が可能

2. コンポーネントの『継承』

■ 標準コンポーネントの『継承』確認

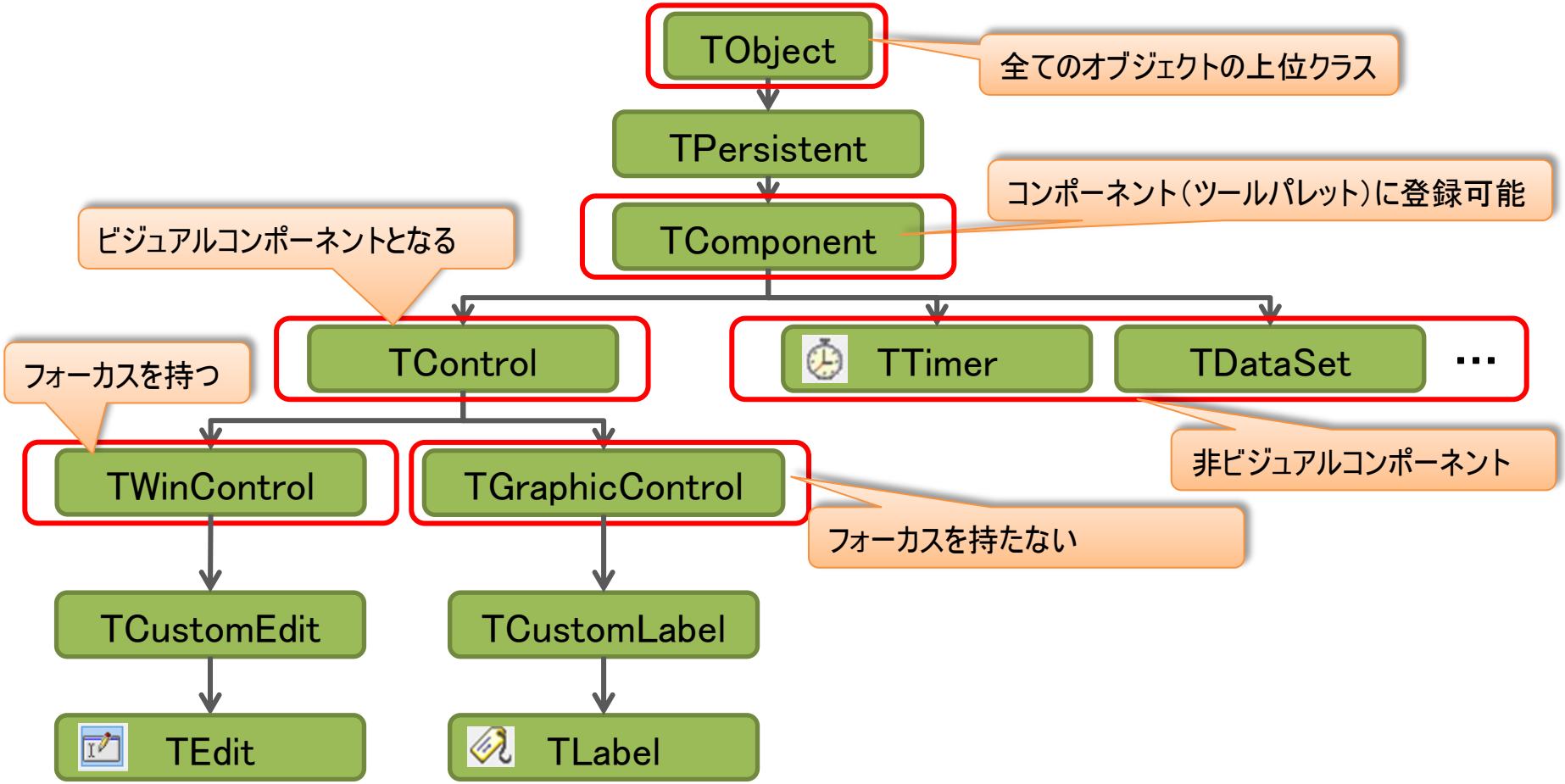
- TEdit をヘルプで確認
  - コンポーネントを選択して[F1]キー押下



コンポーネントは、TObjectを継承して作成されている

■ コンポーネント継承図（一部）

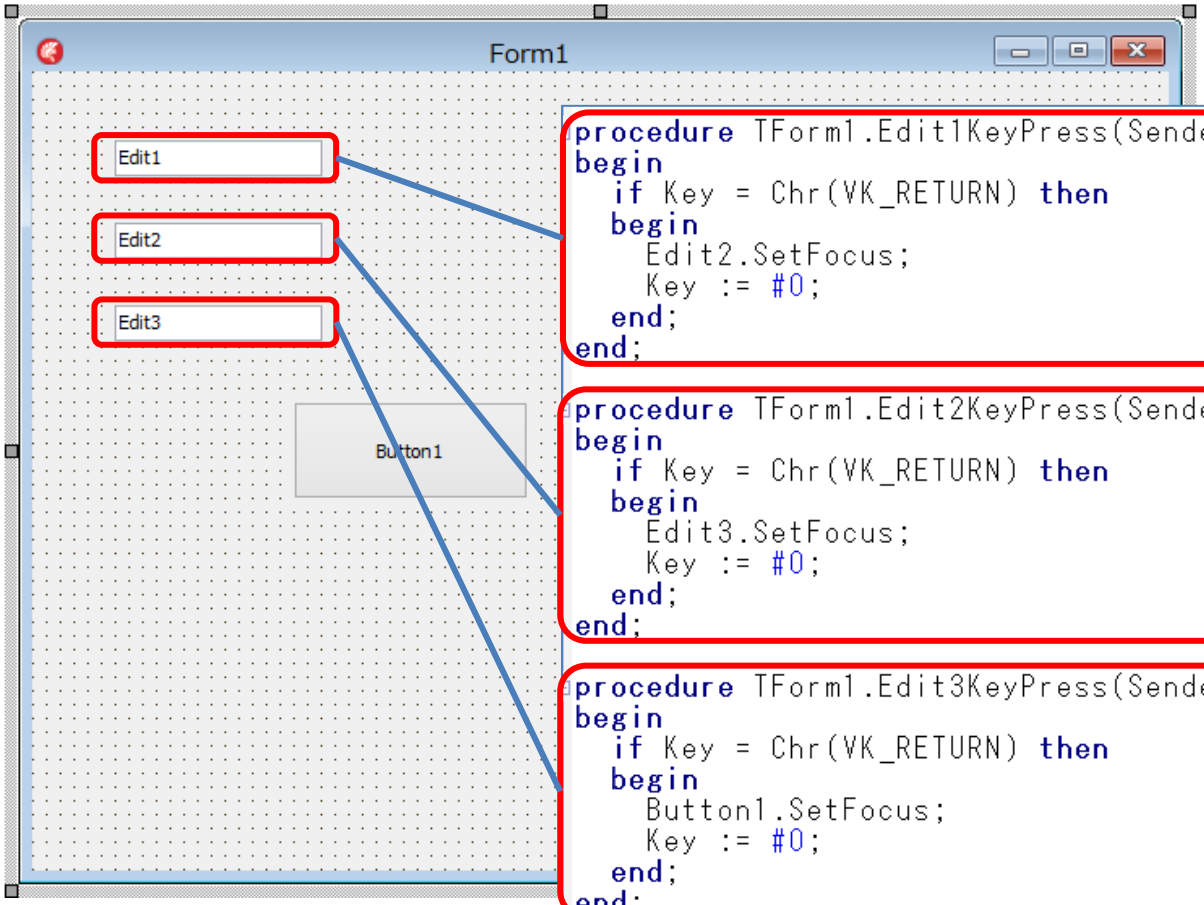
- 全てのコンポーネントは、TObjectを継承して生成



任意のクラスを『継承』することで、機能拡張したコンポーネントが作成可能！

[Enter] キーによる項目移動の実装

- EditコンポーネントのOnKeyPressイベントを使用




```
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
  if Key = Chr(VK_RETURN) then
  begin
    Edit2.SetFocus;
    Key := #0;
  end;
end;
```

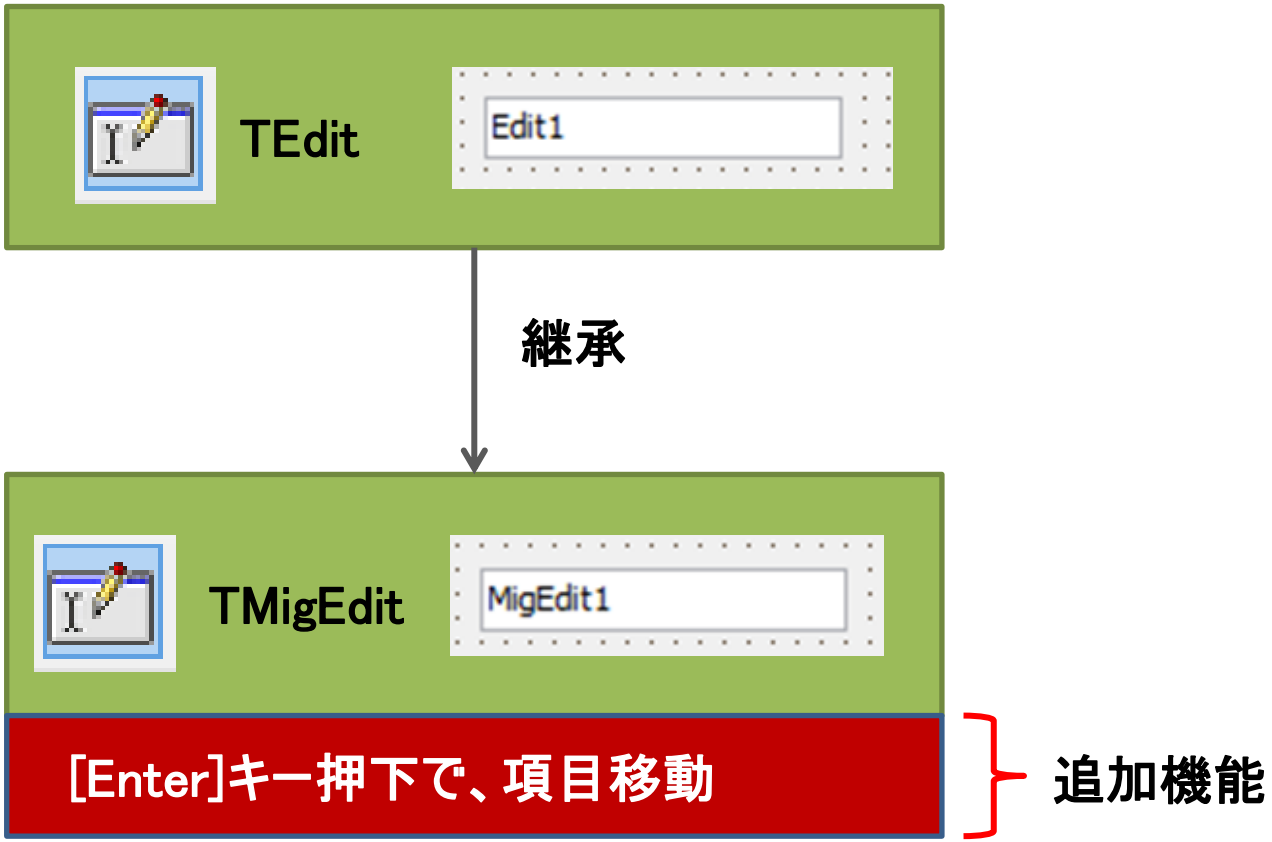
```
procedure TForm1.Edit2KeyPress(Sender: TObject; var Key: Char);
begin
  if Key = Chr(VK_RETURN) then
  begin
    Edit3.SetFocus;
    Key := #0;
  end;
end;
```

```
procedure TForm1.Edit3KeyPress(Sender: TObject; var Key: Char);
begin
  if Key = Chr(VK_RETURN) then
  begin
    Button1.SetFocus;
    Key := #0;
  end;
end;
```

各コンポーネントのイベントに[Enter]キー移動のロジックが必要

■ コンポーネントを『継承』して機能追加

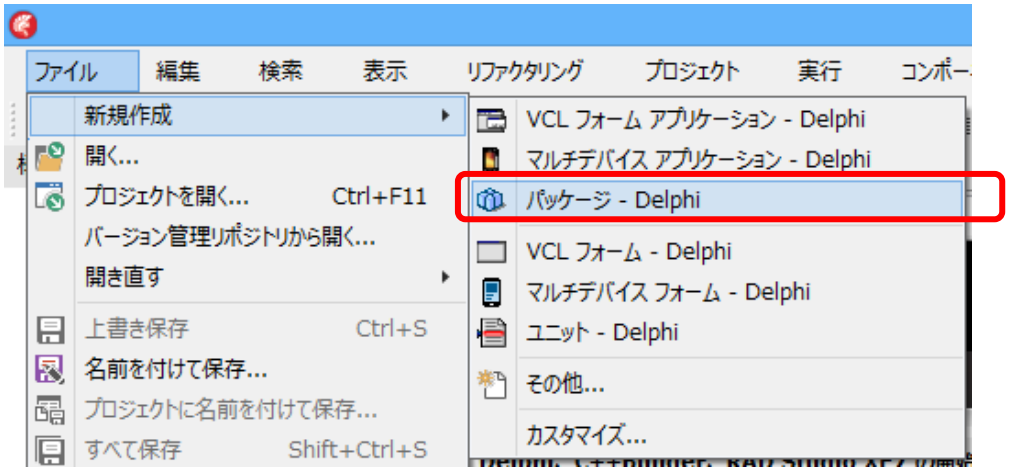
- TEdit　の機能をそのまま継承して、[Enter]キー押下による項目移動を機能追加



■ コンポーネントパッケージの作成

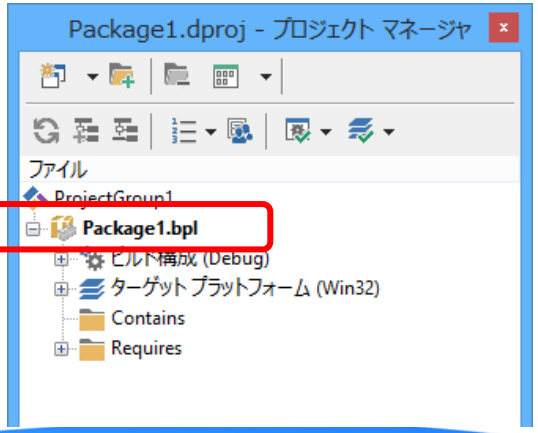
- コンポーネントはパッケージに登録して利用

  [ファイル]→[新規作成]→[パッケージ] を選択



プロジェクトに名前を付けて保存

例： C:¥Projects¥Lib¥MigCompo.bpl

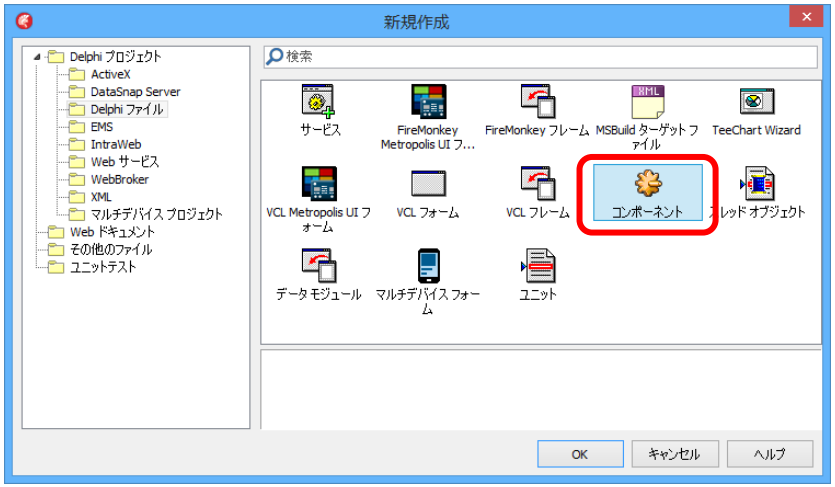


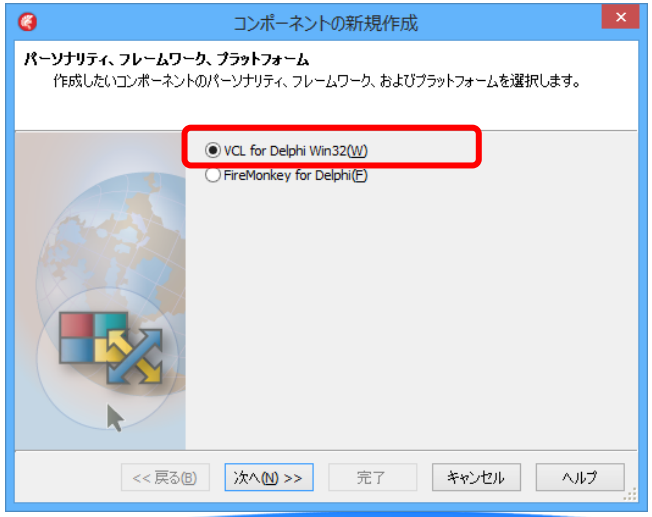
■ 新しいコンポーネントの作成

- 継承元コンポーネントを指定して新しいコンポーネントクラスを作成

  [ファイル]→[新規作成]→[その他] を選択

  新規作成ダイアログより、
  [Delphiファイル]→[コンポーネント]を選択





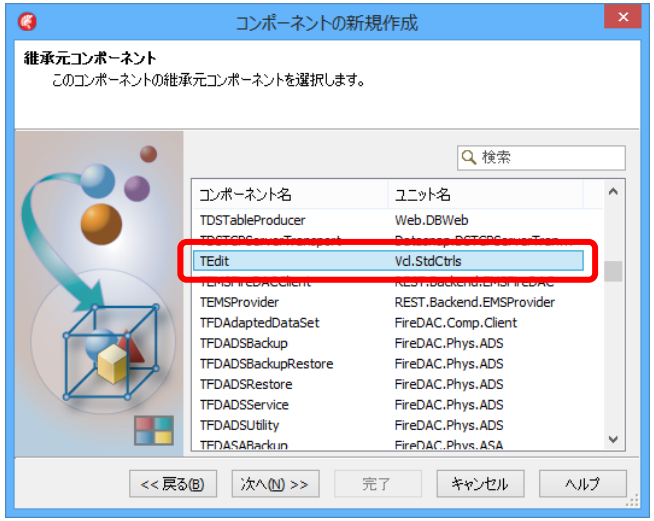
フレームワークを選択（XE3以降）

VCL or FireMonkey

■ 新しいコンポーネントの作成

継承元コンポーネントを指定

TEditを継承



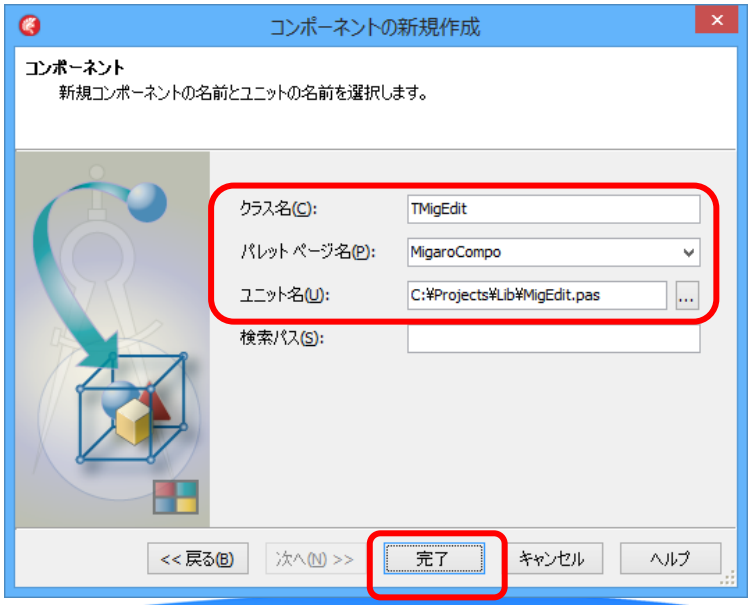
コンポーネントのクラス名を指定

クラス名： TMigEdit

パレットページ名： MigaroCompo

ユニット名：

C:¥Projects¥Lib¥MigEdit.pas



■ ユニットの完成

ウェルカム ページ | MigEdit

```
unit MigEdit;

interface

uses
  System.SysUtils, System.Classes, Vcl.Controls, Vcl.StdCtrls;

type
  TMigEdit = class(TEdit)
  private
    { Private 宣言 }
  protected
    { Protected 宣言 }
  public
    { Public 宣言 }
  published
    { Published 宣言 }
  end;

procedure Register;

implementation

procedure Register;
begin
  RegisterComponents('MigaroCompo', [TMigEdit]);
end;

end.
```

TMigEditクラス　：　TEditクラスを継承

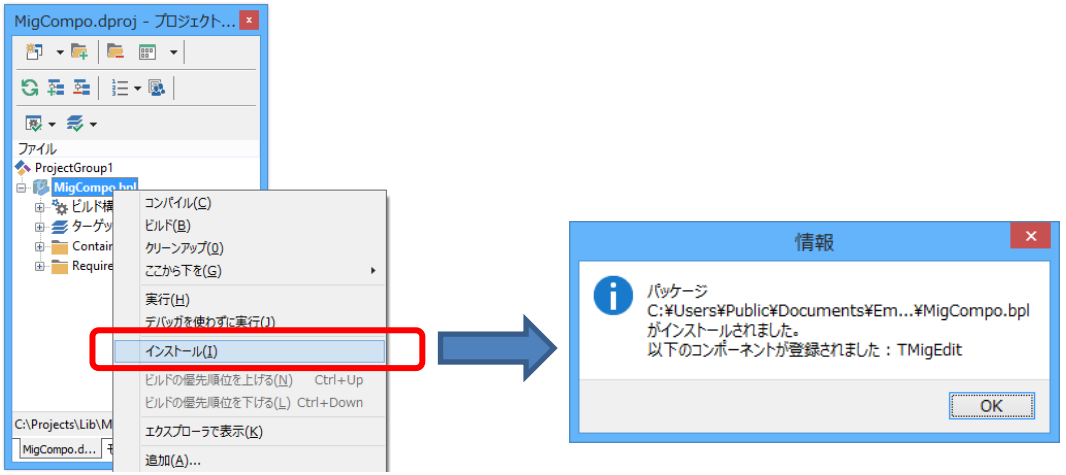
MigaroCompoページにコンポーネントを登録

- クラスの宣言だけで実装（ロジック）は一切なし！

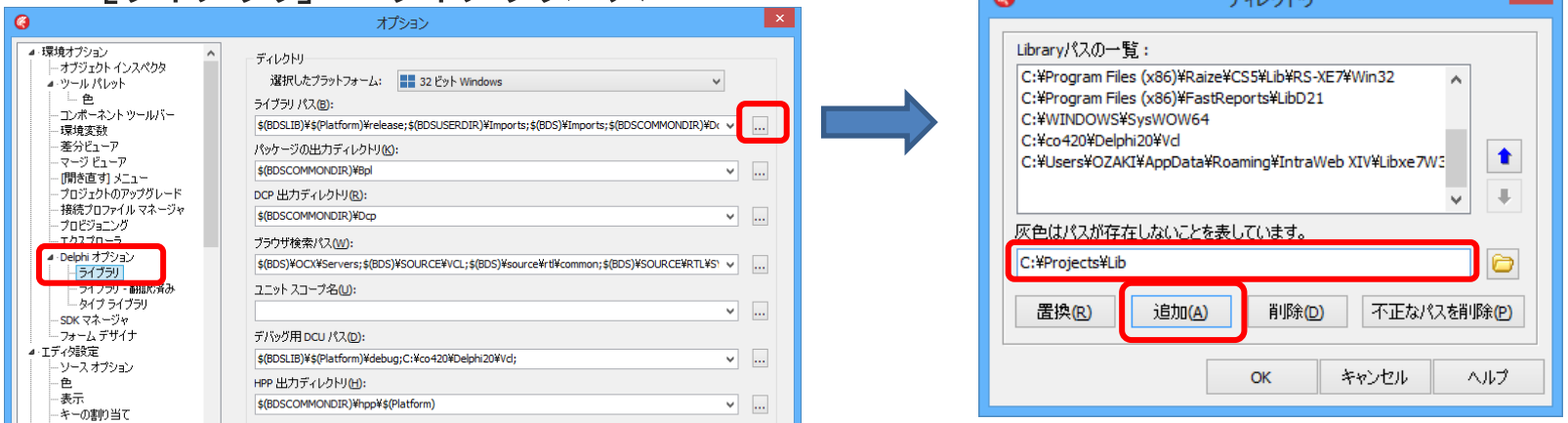
■ コンポーネントパッケージのインストール

- パッケージのインストール及びライブラリパスの追加

パッケージのインストール



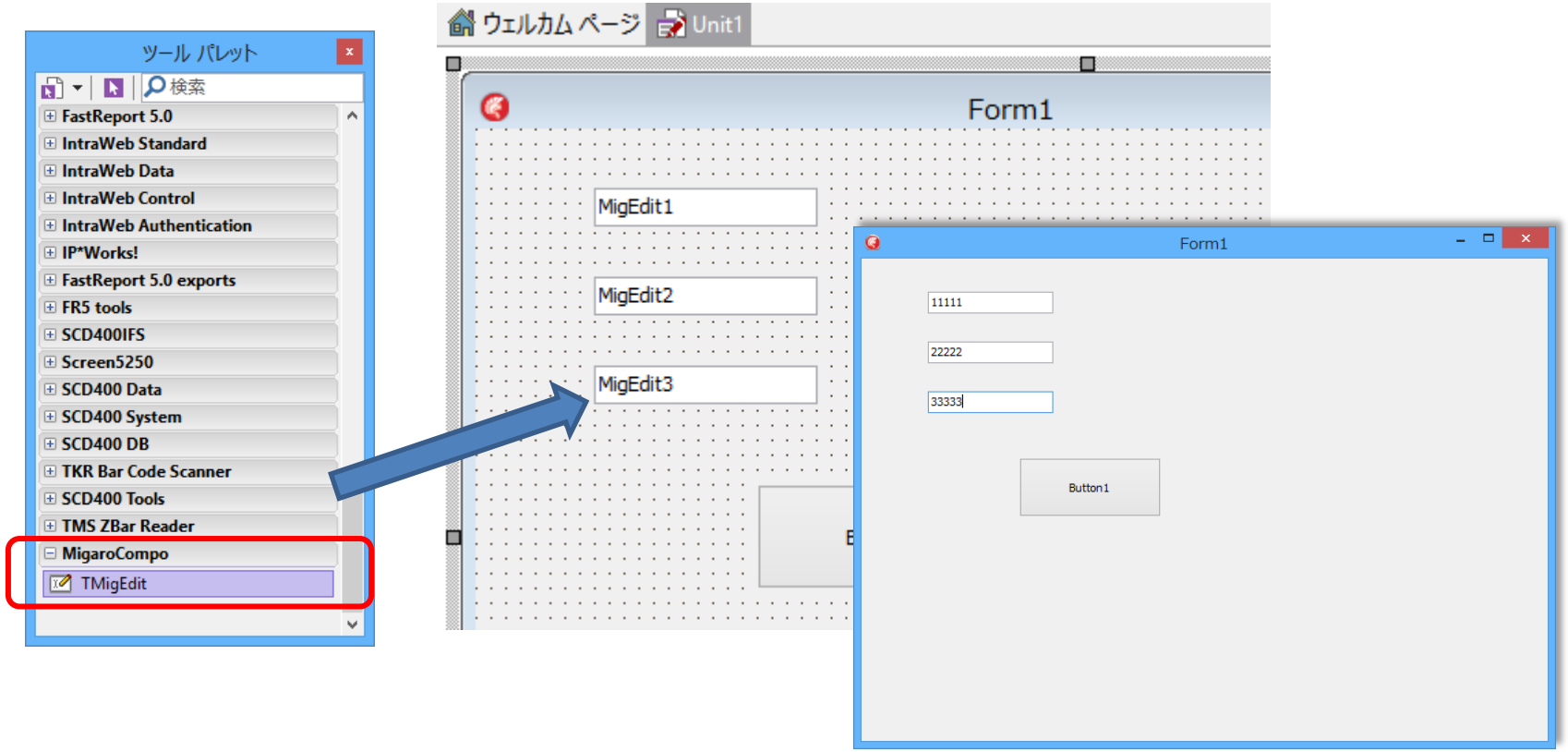
ライブラリパスの追加
[ツール]→[オプション]
→[ライブラリ]→ ライブラリパス



■ 動作の確認

- 新規VCLプロジェクトを作成

  フォームにMigEditコンポーネントを貼り付けて実行



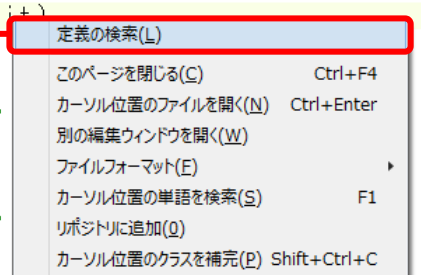
TEditの機能が全て『継承』されている為、TEditと同じ動きとなる！

■ 追加機能の実装

- OnKeyPressイベントの元となる、KeyPressメソッドを継承先でカスタマイズ
  - 継承元コンポーネントは、右クリック→[定義の検索]で確認可能。

    TMigEdit → TEdit → TCustomEdit

```
type
  TMigEdit = class(TEdit)
  private
    { Private 宣言 }
  protected
    { Protected 宣言 }
  public
    { Public 宣言 }
  published
    { Published 宣言 }
  end;
```




```
TCustomEdit = class(TWinControl)
private
  FAlignment: TAlignment;
  FMaxLength: Integer;
  FBorderStyle: TBorderStyle;
  FPasswordChar: Char;
  FReadOnly: Boolean;
  FAutoSize: Boolean;
  FAutoSelect: Boolean;
  FHideSelection: Boolean;
  FOEMConvert: Boolean;
  ⋮
  procedure CMColorChanged(var Message: TMessage); message CM_COLO
  procedure WMContextMenu(var Message: TWMContextMenu); message WM
  procedure WMSetFont(var Message: TWMSetFont); message WM_SETFONT
protected
  function CanObserve(const ID: Integer): Boolean; override;
  procedure ObserverAdded(const ID: Integer; const Observer: IObse
  procedure ObserverToggle(const AObserver: IObserver; const Value
  procedure Change; dynamic;
  procedure KeyDown(var Key: Word; Shift: TShiftState); override;
  procedure KeyPress(var Key: Char); override;
  procedure CreateParams(var Params: TCreateParams); override;
  procedure CreateWindowHandle(const Params: TCreateParams); overr
  ⋮
```

KeyPressメソッドをコピー

■ 追加機能の実装

- 継承先に追加機能を実装

```
type
  TMigEdit = class(TEdit)
  private
    { Private 宣言 }
  protected
    { Protected 宣言 }
    procedure KeyPress(var Key: Char); override;
  public
    { Public 宣言 }
  published
    { Published 宣言 }
  end;
```

override
上位クラスの既存処理の動作を変更すること

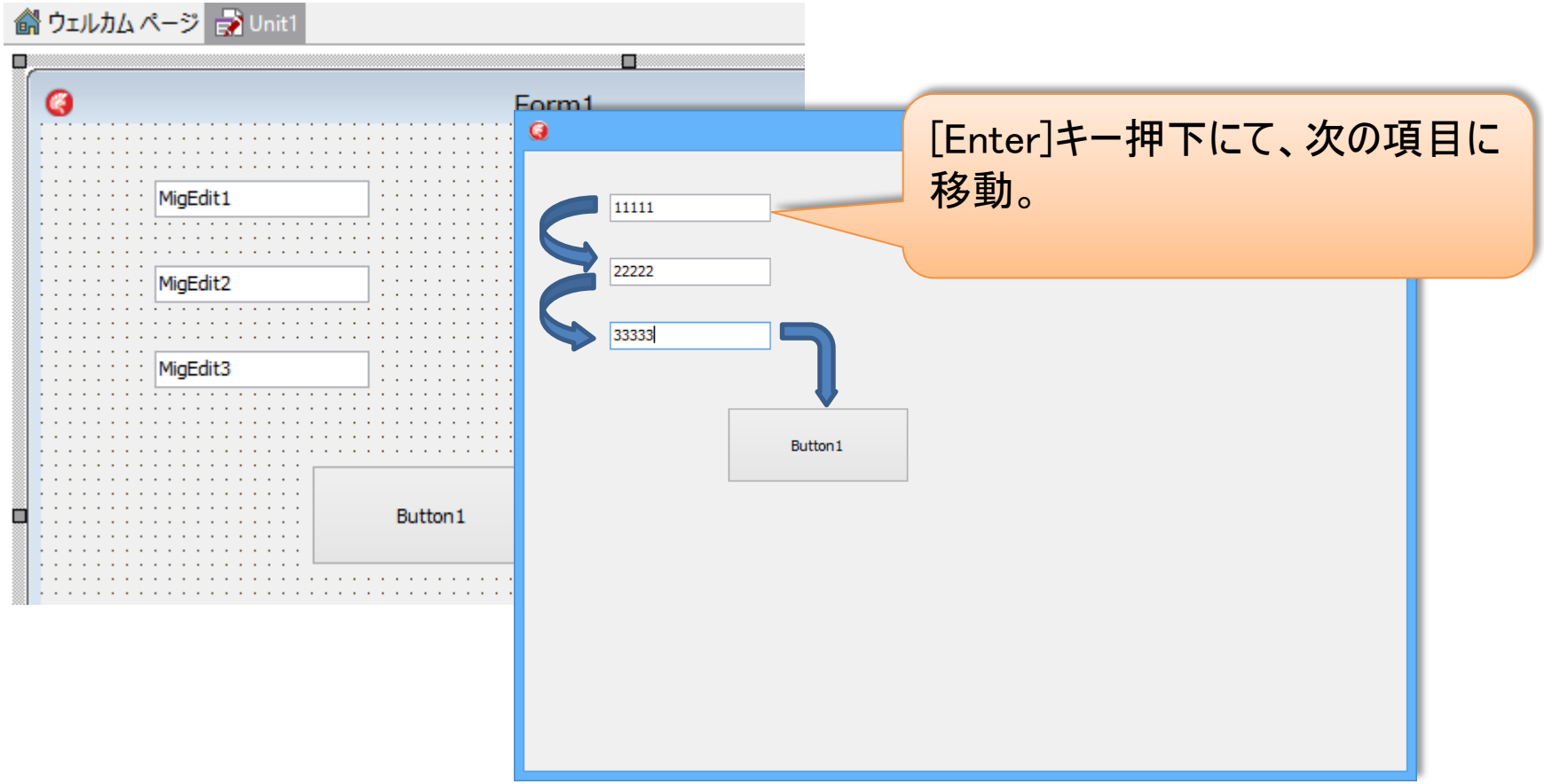
[Ctrl]+[Shift]+[C]

```
{ TMigEdit }

procedure TMigEdit.KeyPress(var Key: Char);
begin
  inherited;

  if Key = Chr(VK_RETURN) then
  begin
    //Tabキー押下
    Keybd_event(VK_TAB, 0, 0, 0);                    //KeyDown
    Keybd_event(VK_TAB, 0, KEYEVENTF_KEYUP, 0);      //KeyUp
    Key := #0;
  end;

end;
```

inherited
継承元のKeyPressメソッドを実行

■ コンポーネント作成後の動作確認

- P.42の確認プログラムを再度実行



プログラムは何も変えていないのに、[Enter]キー項目移動が実現！

■ さらなる改良

- 常に[Enter]キー移動でなく、移動させたくない場合も制御したい。
  - 項目移動処理を行うかどうかのスイッチを追加
  - プロパティとして定義し、オブジェクトインスペクタに表示

```
type
  TMigEdit = class(TEdit)
  private
    { Private 宣言 }
    FEnterNext: Boolean;
  protected
    { Protected 宣言 }
    procedure KeyPress(var Key: Char); override;
  public
    { Public 宣言 }
    constructor Create(AOwner: TComponent); override;
  published
    { Published 宣言 }
    property EnterNext: Boolean read FEnterNext write FEnterNext;
  end;
```

項目移動を行うかどうかの判断フラグ

オブジェクト生成時の初期化イベント

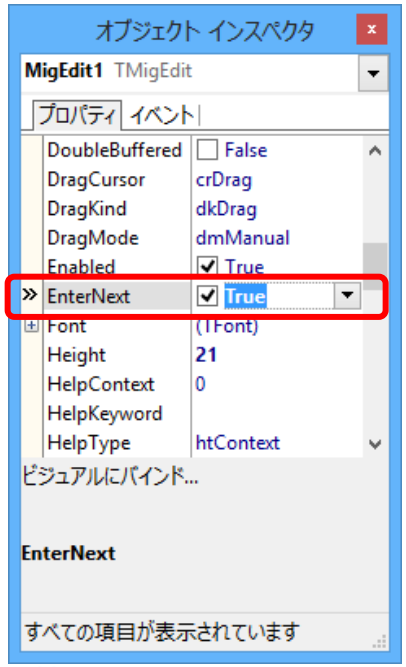
プロパティとして、判断フラグを公開

■ さらなる改良

```
constructor TMigEdit.Create(AOwner: TComponent);
begin
  inherited;
  FEnterNext := True;
end;

procedure TMigEdit.KeyPress(var Key: Char);
begin
  inherited;

  if (Key = Chr(VK_RETURN)) and (FEnterNext) then
  begin
    //Tabキー押下
    Keybd_event(VK_TAB, 0, 0, 0);                    //KeyDown
    Keybd_event(VK_TAB, 0, KEYEVENTF_KEYUP, 0);      //KeyUp
    Key := #0;
  end;

end;
```

初期値は、True（項目移動あり）とする

フラグがTrueの場合のみ実行



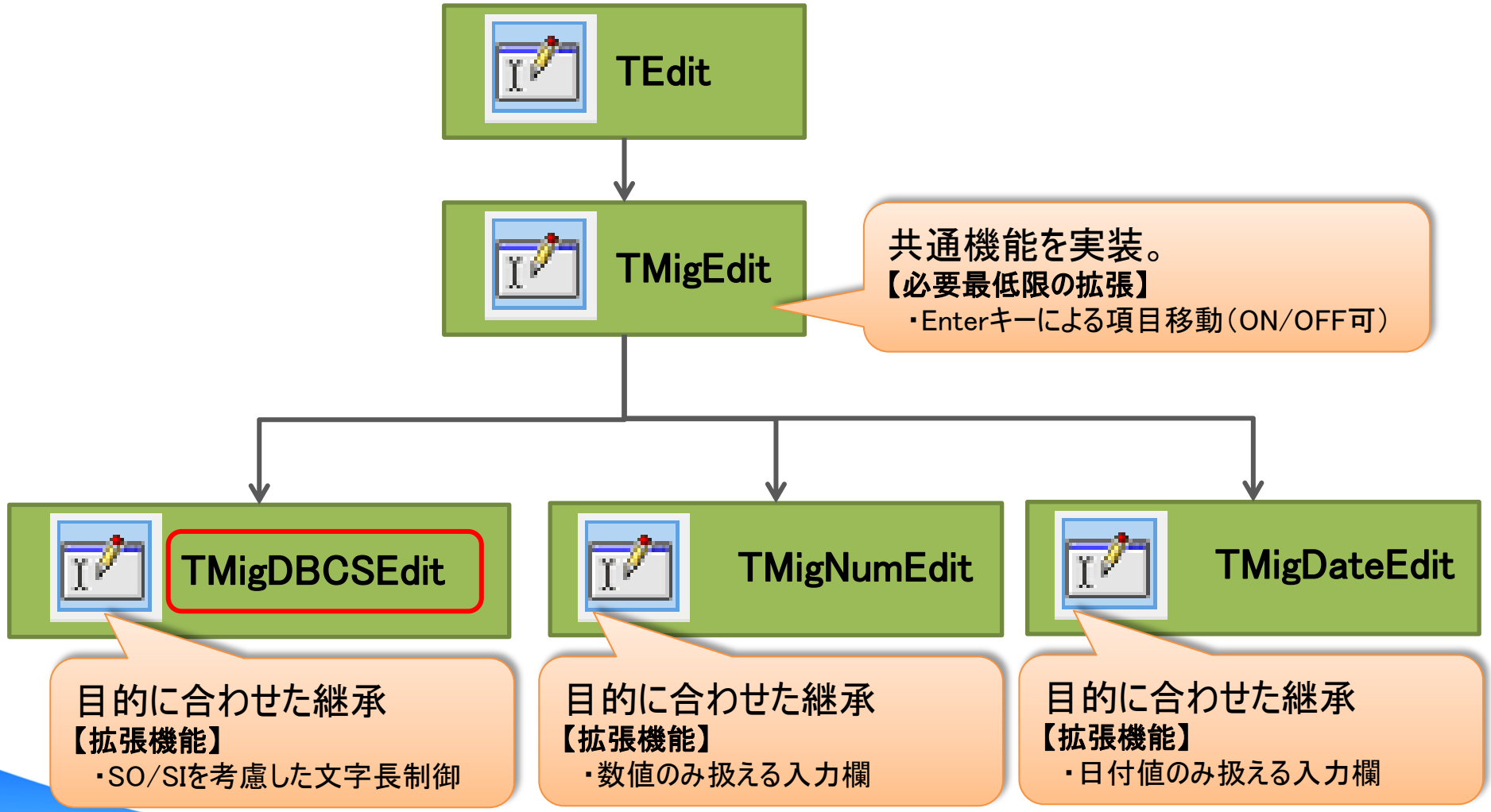
- パッケージを再インストール

  TMigEditに[EnterNext]プロパティが追加

  （Falseに変更すると、Enterキー項目移動しない）

■ コンポーネントの『継承』

- フォーム継承と同様、目的（機能）にあわせた継承が可能

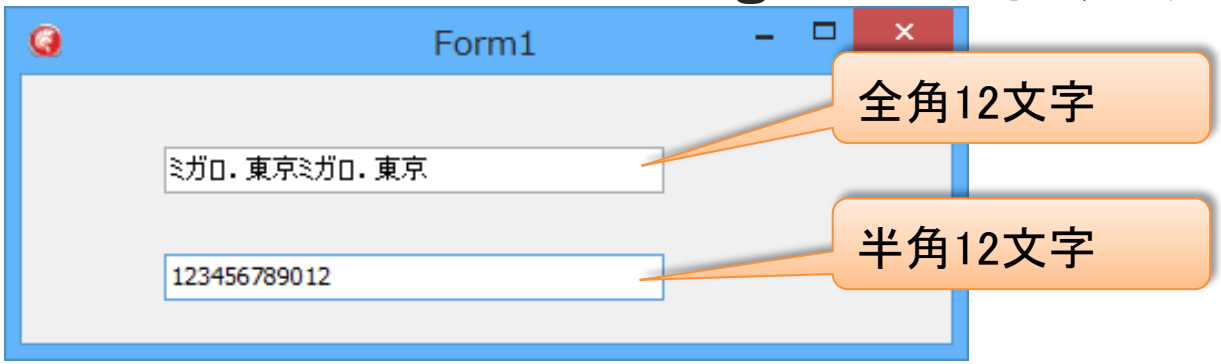


■ IBM i における全角入力について

- 「MIGARO東京」をIBMi上に登録すると…

  - IBMi上では、シフトコード含め12バイト必要

- Delphi/400のTEditでMaxLength=12を指定すると…



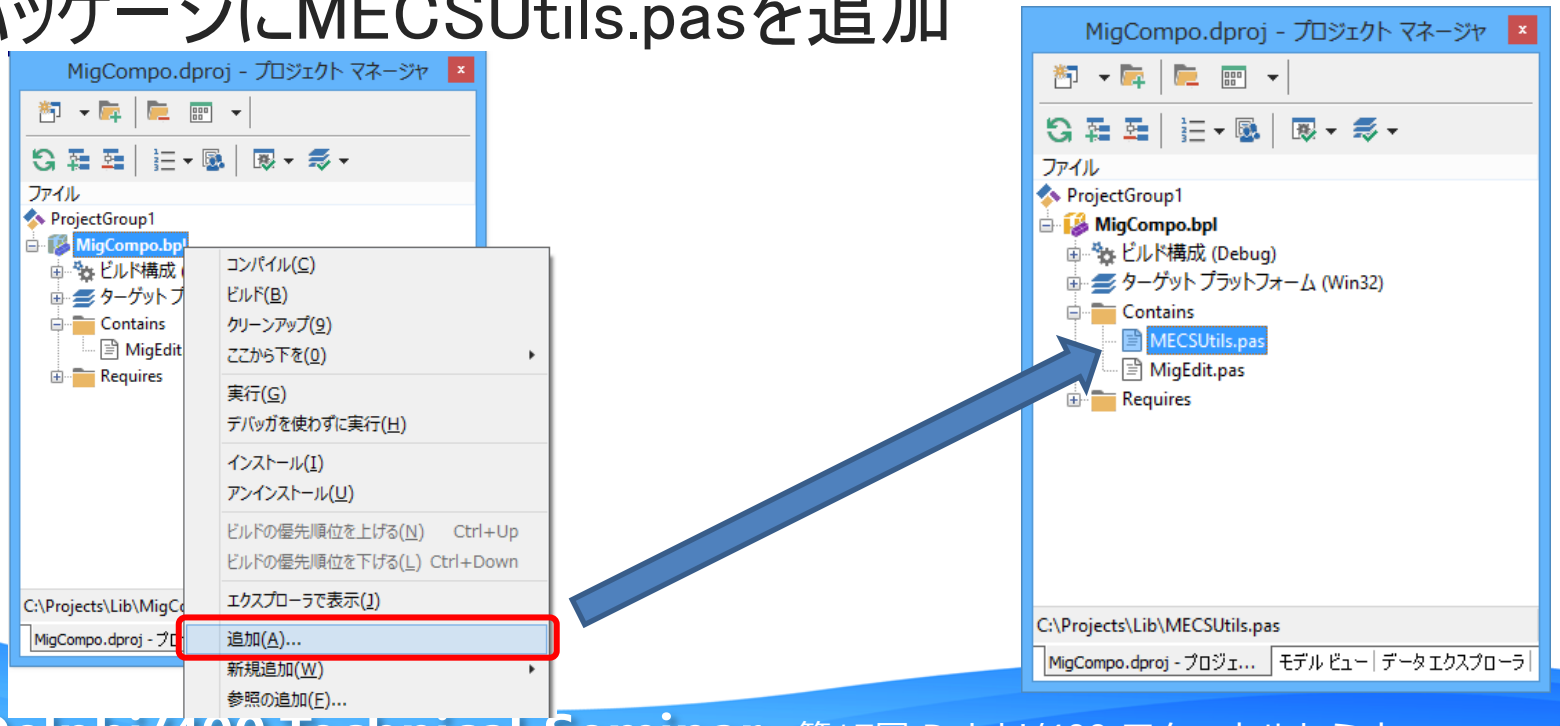
  - V2009以降のDelphi/400では、文字列の取り扱いがUnicodeとなっており、すべての文字を2バイトで表現する為、半角/全角で文字数の差がない。

IBMiのシフト文字を含めたバイト計算で処理できないか？

■ バイト計算用ユニットの追加

- MECSUtils　ライブラリを使用
  - http://cc.embarcadero.com/item/26061　よりダウンロード可能
  - 2015/10/17現在　Ver1.56
  - リファレンス　：　http://ht-deko.com/tech021.html

- パッケージにMECSUtils.pasを追加



■ シフト文字を考慮した文字列バイト数取得関数

```
function EbcdicLength(s: String): Integer;
var
  i: Integer;
  inMBCS, ckMBCS: Boolean;
begin
  // 初期値セット
  Result := 0;
  // シフトコード内部フラグ初期化
  inMBCS := False;

  for i := 1 to MecsLength(s) do
  begin
    // 全角(FullWidth)文字判定
    ckMBCS := MecsIsFullWidth(s, i);
    // 文字長計算
    if ckMBCS then
      Result := Result + 2
    else
      Result := Result + 1;
    // シフトコード計算(半角→全角切替時に開始/終了コード分追加)
    if ckMBCS and not inMBCS then
    begin
      Result := Result + 2;
      inMBCS := True;
    end
    else
      if not ckMBCS then
        inMBCS := False;
  end;
end;
```

s：チェック対象　文字列

s ＝“MIGARO東京”をセットすると
Result＝12　が返却

■ 指定バイト数分の文字列抜き出し処理

AText：チェック対象　文字列
AMaxLength：　シフト文字を含むバイト数

```
function GetLengthText(AText: String; AMaxLength: Integer): String;
begin
  if (AMaxLength <= 0) or (EbcdicLength(AText) <= AMaxLength) then
    Result := AText
  else
  begin
    repeat
      AText := MecsCopy(AText, 1, MecsLength(AText) - 1);
    until (EbcdicLength(AText) <= AMaxLength);
    Result := AText;
  end;
end;
```

AMaxLength = 12 を指定した場合

- AText　=“MIGARO東京”をセット　→　Result =“MIGARO東京”(12バイト)
- AText =“ミガロ. 東京ミガロ. 東京”をセット→Result=“ミガロ. 東”(12バイト)
- Atext =“123456789012”をセット　→　Result =“123456789012”(12バイト)

コンポーネントにこの計算処理を組み込めば良い！

■ コンポーネントへの組み込み

- 文字変更イベントであるOnChangeイベントの元となる、Changeメソッドを継承先でカスタマイズ

```
//DBCSコンポーネント
TMigDBCSEdit = class(TMigEdit)
private
  { Private 宣言 }
protected
  { Protected 宣言 }
  procedure Change; override;
public
  { Public 宣言 }
  constructor Create(AOwner: TComponent); override;
published
  { Published 宣言 }
end;
```

Edit内の文字列が変更されるときに、シフト文字を考慮した長さに調整

■ コンポーネントへの組み込み

- 元の文字列とシフト文字考慮した文字列とが異なる場合、置き換えを行う

```
{ TMigDBCSEdit }

procedure TMigDBCSEdit.Change;
var
  iSelStart: Integer;
  sWKStr: String;
  iOrdLength: Integer;
begin
  inherited;

  //デザイン時は何もしない
  if csDesigning in ComponentState then Exit;

  //初期化
  iSelStart := SelStart;
  iOrdLength := Length(Text);

  //最大桁数のチェック
  if MaxLength <> 0 then
  begin
    sWKStr := GetLengthText(Text, MaxLength);

    if Text <> sWKStr then
    begin
      Text := sWKStr;
      SelStart := iSelStart - (iOrdLength - Length(sWKStr));
    end;
  end;
end;
```

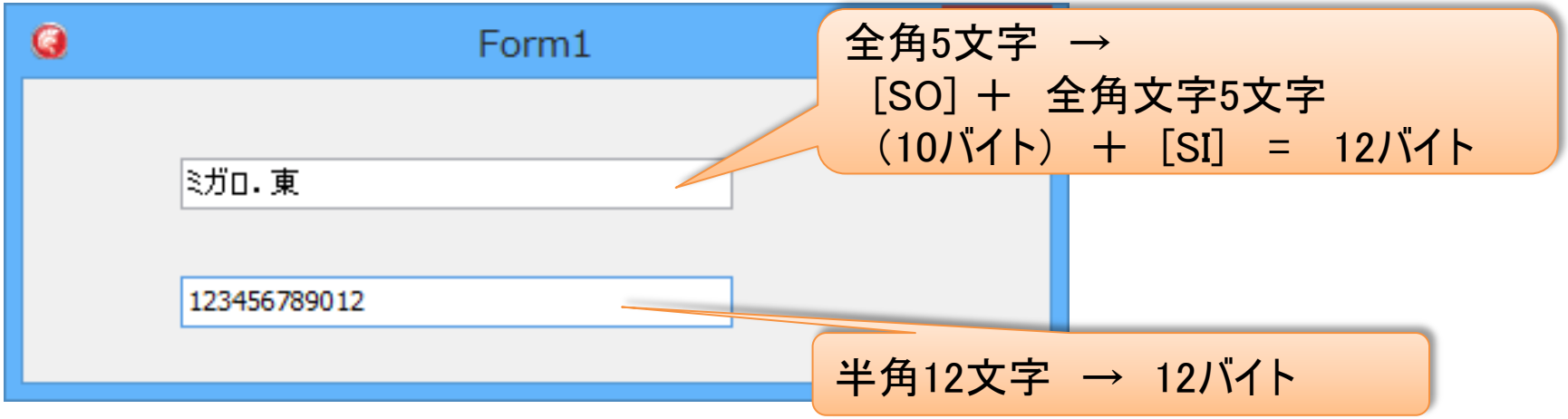
画面設計時は、制御せずプログラム実行時のみ制御する。

シフト文字を考慮したMaxLength分の文字列を取得。

元のTextと取得値が異なる場合置き換え

■ 実行例

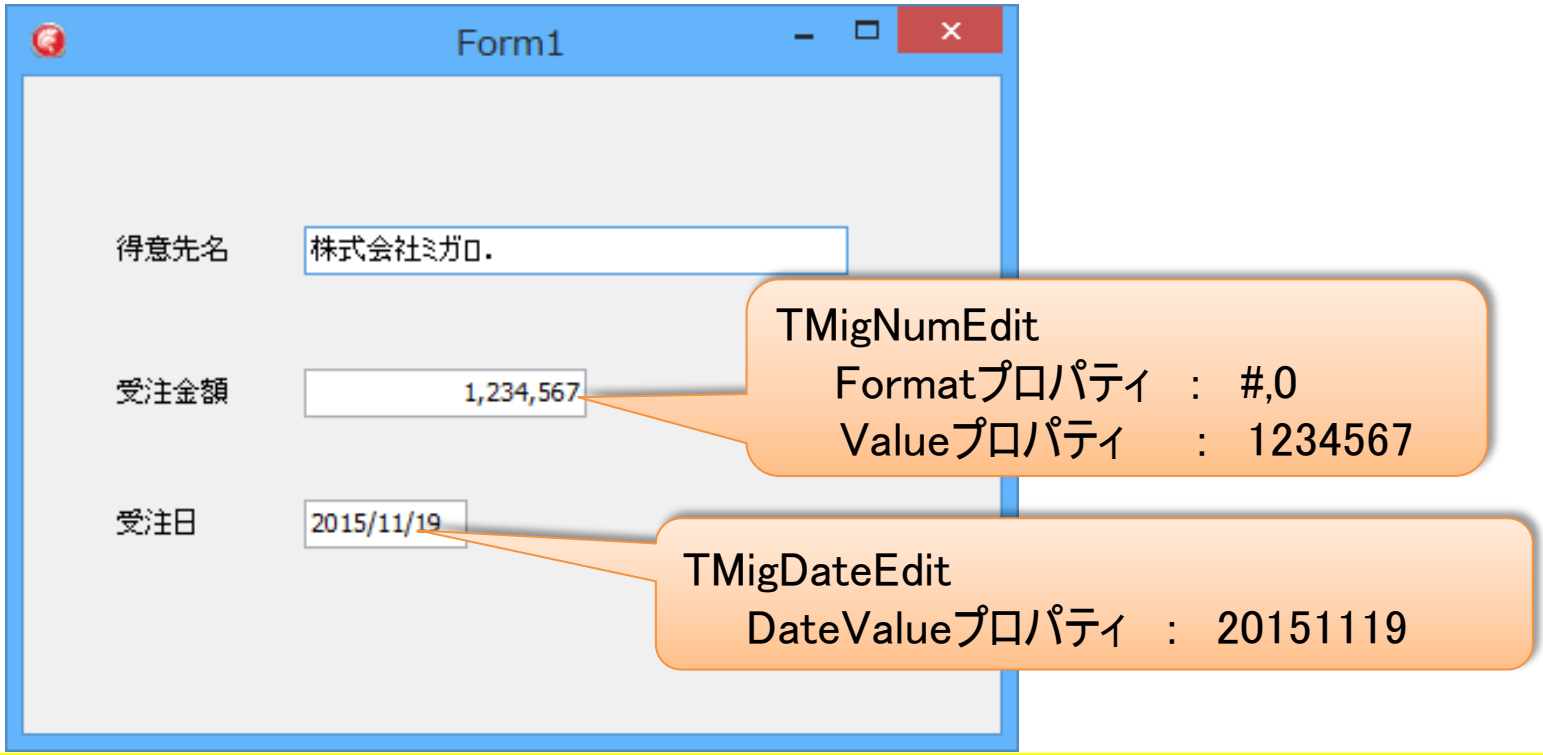
- MaxLength=12を指定



MaxLengthの指定だけで、IBMiの文字長制御が可能！

■ コンポーネント拡張例

- 数値入力エディット、日付値入力エディット
  - TMigNumEdit(数値用)、TMigDateEdit(日付値用)
  - ソースはサンプルCDに収録



TMigEditを継承しているので、全てEnterキー項目移動が可能

■ コンポーネントの『継承』

- コンポーネントは、全てTObjectを起点とする階層構成となっている。
- コンポーネントを継承することで、独自の機能を追加可能。
- コンポーネントも目的別に継承すると効果的に拡張可能

【補足】
TObjectを『継承』した
業務ロジック一元化

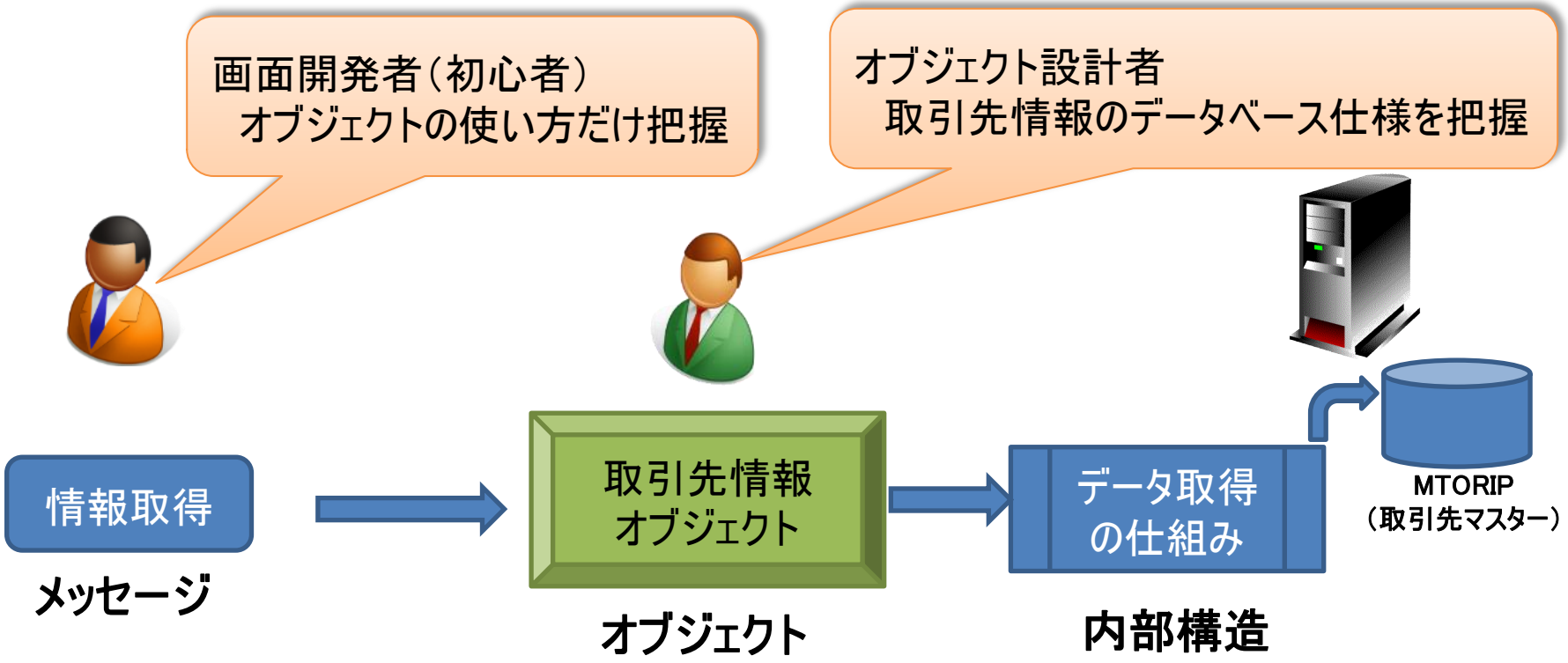
TObjectについて

- Delphi/400のすべてのオブジェクトの継承元
  - TObject
    - フォーム
    - コンポーネント

- TObjectを『継承』することで、あらゆる処理をオブジェクト化可能
  - オブジェクト化することにより、オブジェクト利用者はオブジェクト内部処理を知らなくても使用可能となる。
  - データと処理を一つのオブジェクトにまとめることが出来る。

TObjectを『継承』することで、業務ロジックをオブジェクト化可能

■ 取引先情報取得処理のオブジェクト化



■ オブジェクトプログラム例

- 宣言部

ウェルカム ページ | TorihikiGetFrm | pasDataAccess

```
type
  //取引先情報クラス
  TTorihikiClass = class(TObject)
  private
    //内部変数（フィールド）
    FToriNo: Integer;      //取引先No
    FToriName: String;     //取引先名
    FKubun: String;        //取引先区分
    FAddress1: String;     //住所１
    FAddress2: String;     //住所２
    FTel: String;          //TEL
    FFax: String;          //FAX
  public
    //コンストラクター
    constructor Create(ANo: Integer);
    //アクセス用プロパティ
    property ToriNo: Integer  read FToriNo;    // 取引先No
    property ToriName: String read FToriName;  // 取引先名
    property Kubun: String    read FKubun;     // 取引先区分
    property Address1: String read FAddress1;  // 住所１
    property Address2: String read FAddress2;  // 住所２
    property Tel: String      read FTel;       // TEL
    property Fax: String      read FFax;       // FAX
  end;
```

TObjectを『継承』して独自クラスを作成

オブジェクト内部で使用する変数

公開される
プロパティ(読取専用)と
メソッド

■ オブジェクトプログラム例

- 実装部
  - データベース仕様に基づき、オブジェクト生成時にIBMiよりデータを取得

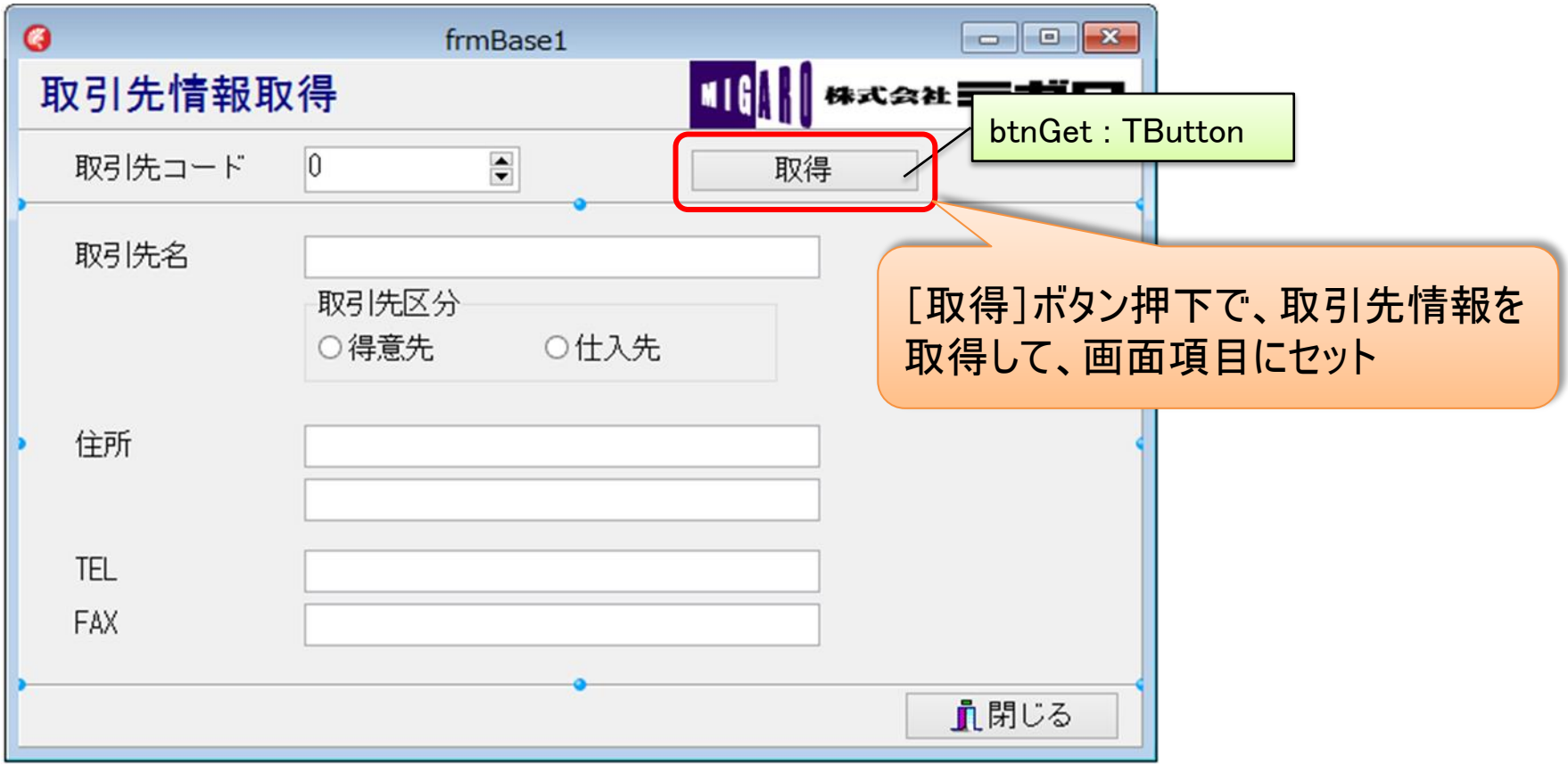
```
{ TTorihikiClass }

uses MainDM;

constructor TTorihikiClass.Create(ANo: Integer);
begin
  with dmMain.qryTemp do
  begin
    SQL.Clear;
    SQL.Add('SELECT * FROM MTORIP WHERE TOTRNO = :TRNO');
    ParamByName('TRNO').AsInteger := ANo;
    Active := True;
    try
      if Eof and Bof then
        raise Exception.Create('取引先データは存在しません');

      FToriNo := FieldByName('TOTRNO').AsInteger;    // 取引先No
      FToriName := FieldByName('TOTRNM').AsString;   // 取引先名
      FKubun := FieldByName('TOTRKB').AsString;      // 取引先区分
      FAddress1 := FieldByName('TOADR1').AsString;   // 住所１
      FAddress2 := FieldByName('TOADR2').AsString;   // 住所２
      FTel := FieldByName('TOTEL').AsString;         // TEL
      FFax := FieldByName('TOFAX').AsString;         // FAX
    finally
      Active := False;
    end;
  end;
end;
```

取引先Noをパラメータとしたオブジェクトを生成処理

データベースフィールドを内部変数に転送

■ オブジェクトを使用したプログラムの開発

- 画面レイアウト例



■ 画面プログラム例

```
uses pasDataAccess;

procedure TfrmBase1.btnGetClick(Sender: TObject);
var
  TorihikiData: TTorihikiClass;
begin
  inherited;
  //取引先オブジェクトの生成
  TorihikiData := TTorihikiClass.Create(sedTRNO.Value);
  try
    //画面に値をセット
    edtTorihikiName.Text := TorihikiData.ToriName;
    rgTorikubun.ItemIndex := StrToInt(TorihikiData.Kubun) - 1;
    edtAddress1.Text := TorihikiData.Address1;
    edtAddress2.Text := TorihikiData.Address2;
    edtTel.Text := TorihikiData.Tel;
    edtFax.Text := TorihikiData.Fax;
  finally
    //オブジェクトの破棄
    TorihikiData.Free;
  end;
end;
```

取引先Noをパラメータとして、オブジェクトを生成

オブジェクトのプロパティより情報取得

IBMiデータベースに関する情報は一切コードに含まれない（隠蔽化）

■ 業務ロジックのオブジェクト化メリット

- 作業の分割
  - データベース仕様やロジックを検討するオブジェクト開発者と、画面を作成する開発者とが個別に開発可能。
- 機能の隠蔽化
  - 画面開発者は、データベースアクセス仕様を知らなくても、オブジェクトの使用方法が分かればプログラム開発が可能。
- 処理の一元化
  - データベース処理（ロジック）がオブジェクトで一元管理できるため、仕様変更や機能拡張が行いやすい。

まとめ

■ まとめ

- 『継承』を使用した開発効率化を図る手法を2つ紹介
  - フォームの『継承』
    - フォームを継承することにより、画面のタイプごとに開発手法を統一化
      - 継承画面の作成方法
      - 一覧照会画面継承例
      - 継承元画面の機能拡張
  - コンポーネントの『継承』
    - コンポーネントを継承することにより、汎用的な処理を部品化
      - コンポーネントの作成手順
      - Enterキー制御の追加
      - プロパティ追加方法
      - 継承によるコンポーネントの目的に応じた機能拡張

ご清聴ありがとうございました。