【セッションNo．5】

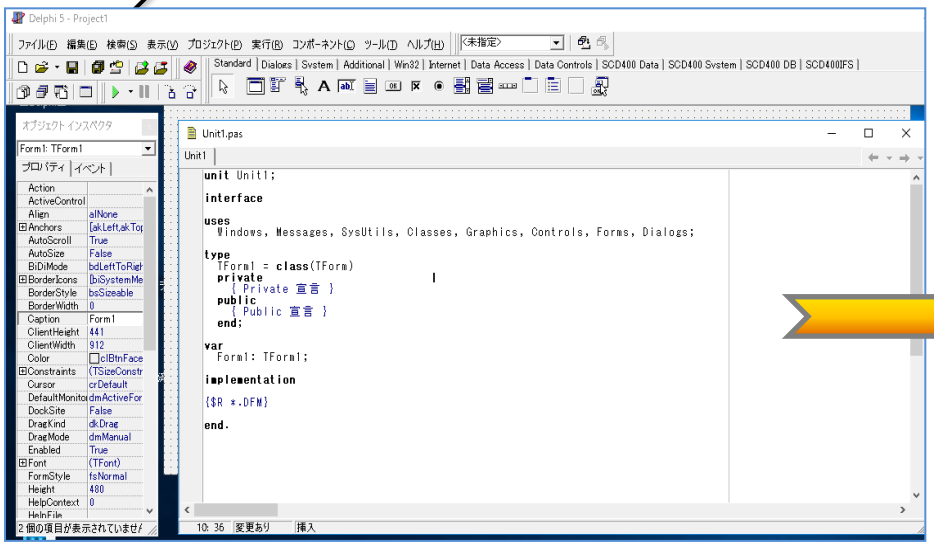
# 徹底活用！Delphi/400開発環境TIPS

株式会社ミガロ.
RAD事業部　営業・営業推進課
尾崎　浩司

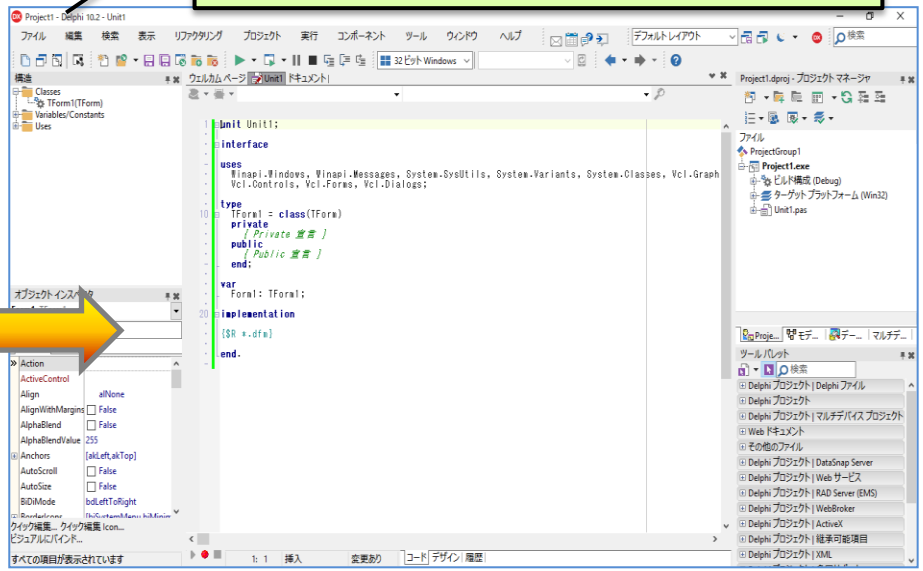
# ■ はじめに

- Delphi/400はバージョンアップに伴い、マルチデバイス対応やコンポーネントの機能強化だけでなく、プログラム開発を行うIDE（統合開発環境)自体も進化しています。
本セッションでは、知っていると開発時のプログラミング効率が大きく向上するIDE機能や、チーム開発で役立つソース管理ツールとの連携についてTIPSをご紹介致します。

**Delphi/400 V5 のIDE**




**Delphi/400 10.2 のIDE**
基本構成は従来のバージョンと変わらないが、機能は大きく進化している



# 【アジェンダ】

1．ソース開発機能

- 1-1.コード支援機能
- 1-2.ライブテンプレート
- 1-3.リファクタリング
- 1-4.ソース整形

2．ソース管理機能

- 2-1.ソース管理ツール
- 2-2.環境構築
- 2-3.使用方法

3．まとめ

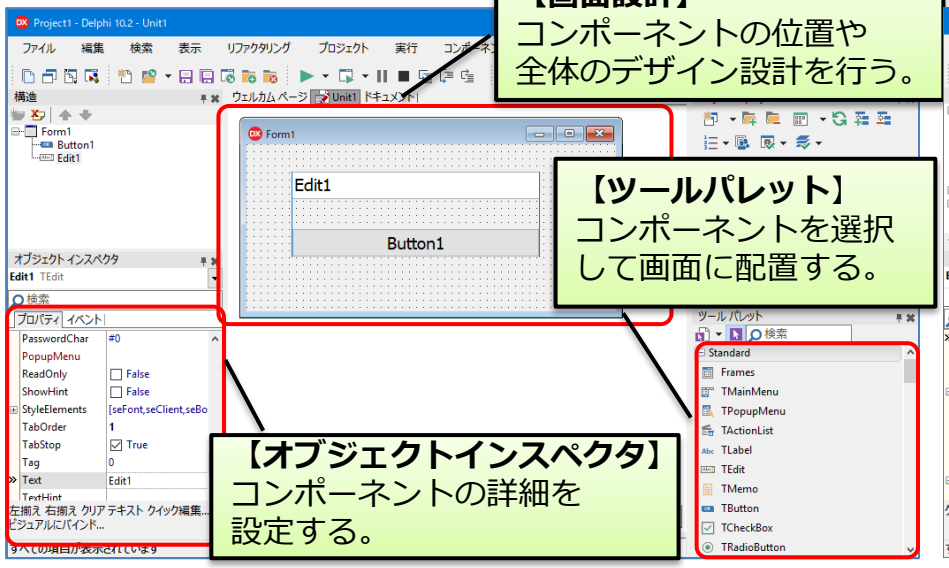
# １．ソース開発機能

# 1.ソース開発機能

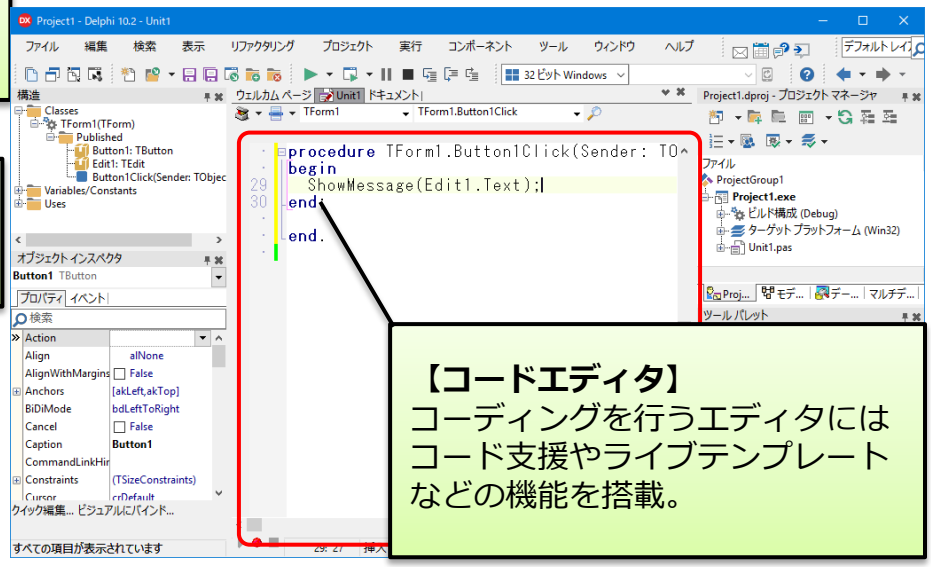
- Delphi/400 IDE（統合開発環境）の概要

  Delphi/400のIDEはアプリケーション開発で必要となる
  画面設計からコーディングまでの全ての機能を搭載しており、
  1ツールで開発、コンパイル、デバッグが行える環境。

【画面設計時】




【コーディング時】




時間がかかるソース開発・メンテナンスの効率が向上する機能

- コード支援機能
- リファクタリング機能
- ライブテンプレート機能
- ソース整形機能

# 1-1.コード支援機能

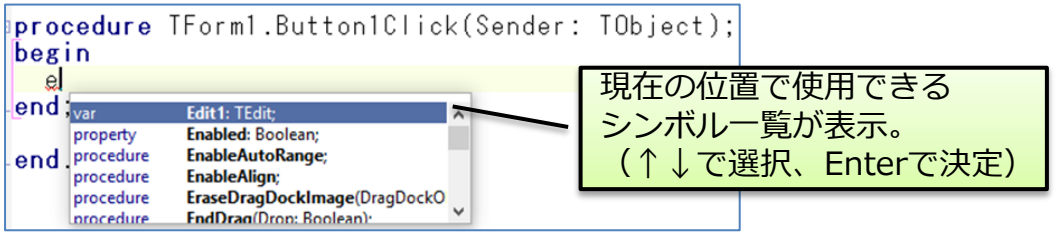
## コード支援機能

- コード入力時にコンポーネント名やプロパティ、メソッドなど
  コーディング中に予想されるコードを自動補完してくれる。

＜役立つ機能＞

①コード補完　**Ctrl＋Space**

- コードを入力する際、全て
  打ち込まなくても、選択肢
  から自動入力できる。



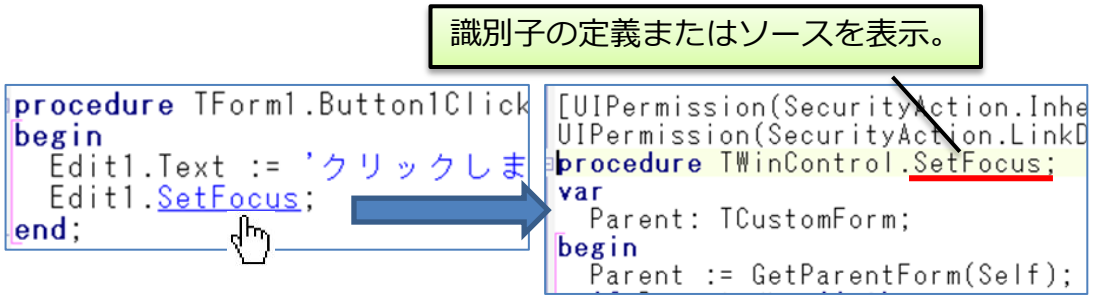

②コードパラメータヒント　**Ctrl＋Shift＋Space**

- メソッド呼び出しの引数の
  名前と型を確認できる。

# 1-1.コード支援機能

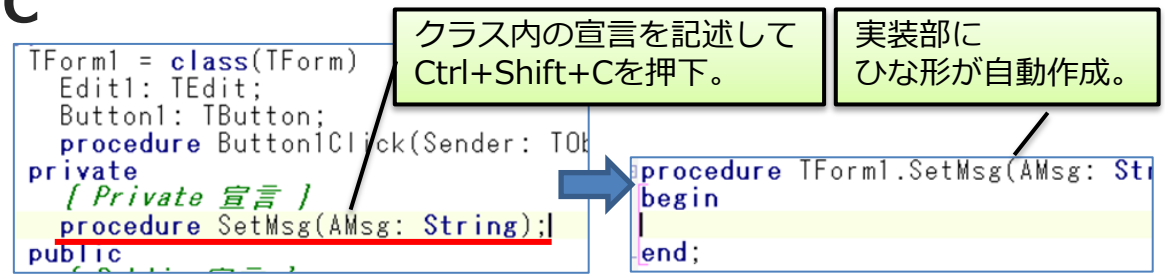
## ③コード参照　Ctrl+クリック

- 任意のクラス、変数、メソッド プロパティ等の宣言やソースを参照できる。



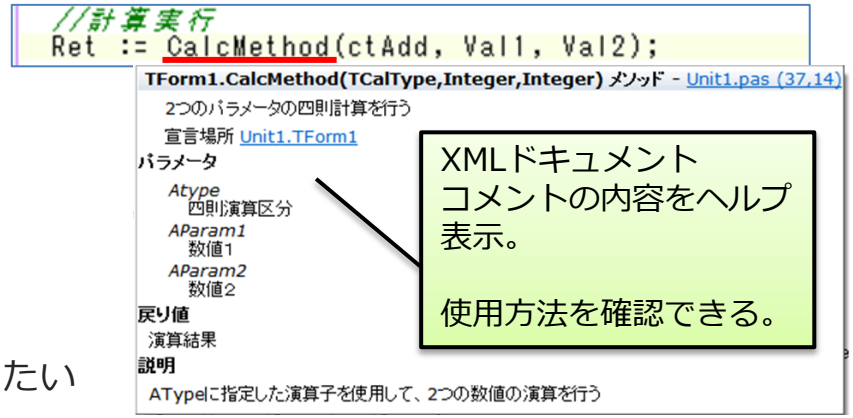

## ④クラス補完　Ctrl+Shift+C

- メソッドやプロパティ宣言をもとに、フィールドやメソッドの実装部ひな形が自動作成される。




## ⑤ヘルプインサイト　Ctrl+Shift+H

- メソッド等の宣言元のユニットやパラメータ等の情報をヘルプ表示する。
- XMLドキュメントコメントを作成すれば独自のヘルプも表示できる。
  ※　XMLドキュメントコメントについては、第21回テクニカルセミナー「開発者が知りたい実践プログラミングテクニック」参照



# 1-2.ライブテンプレート機能

## ライブテンプレート機能

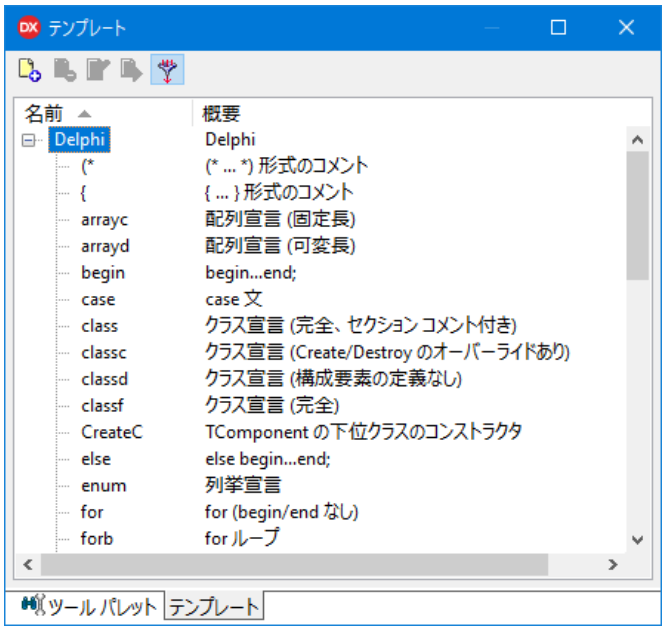
- コーディング上でよく使用するif文やfor文、try-finally-end といった構文を自動補完してくれる。
  コード支援機能と異なり、ロジックなどの構文が利用できる。

＜役立つ機能＞

- コードエディタ上では、Ctrl+J 押下で テンプレートメニューを表示できる。

- 登録済みの標準テンプレートは、 [表示|テンプレート]で確認できる。

- テンプレートは、独自テンプレートとして 業務ロジックを登録することもできる。



# 1-2.ライブテンプレート機能

## <ライブテンプレートの使用方法>

（例）“try”：オブジェクト生成/破棄ロジックを追加

```
procedure TForm1.Button1Click(Sender: TObject);
var
  bm: TBitmap;
begin
  try

end;
```

テンプレートの登録ワードを入力し、Tabを押下すると定義済みコードが挿入。

```
procedure TForm1.Button1Click(Sender: TObject);
var
  bm: TBitmap;
begin
  MyClass := TComponent.Create(Self);
  try

  finally
    MyClass.Free;
  end;

end;
```

四角枠（ジャンプポイント）は、Tab / Shift-Tabで移動。

```
procedure TForm1.Button1Click(Sender: TObject);
var
  bm: TBitmap;
begin
  bm := TComponent.Create(Self);
  try

  finally
    bm.Free;
  end;

end;
```

変数部分を入力すると関連部分も自動的に反映。

クラス名は、自動的にコード補完が表示。

```
procedure TForm1.Button1Click(Sender: TObject);
var
  bm: TBitmap;
begin
  bm := Tbitm.Create(Self);
  try

  finally
    bm.Fre
  end;

end;
```

```
type TBitmapImage: class(TSharedImage);
type TBitmap: class(TGraphic);
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
  bm: TBitmap;
begin
  bm := TBitmap.Create;
  try

  finally
    bm.Free;
  end;

end;
```

ジャンプポイントを抜けると完成。

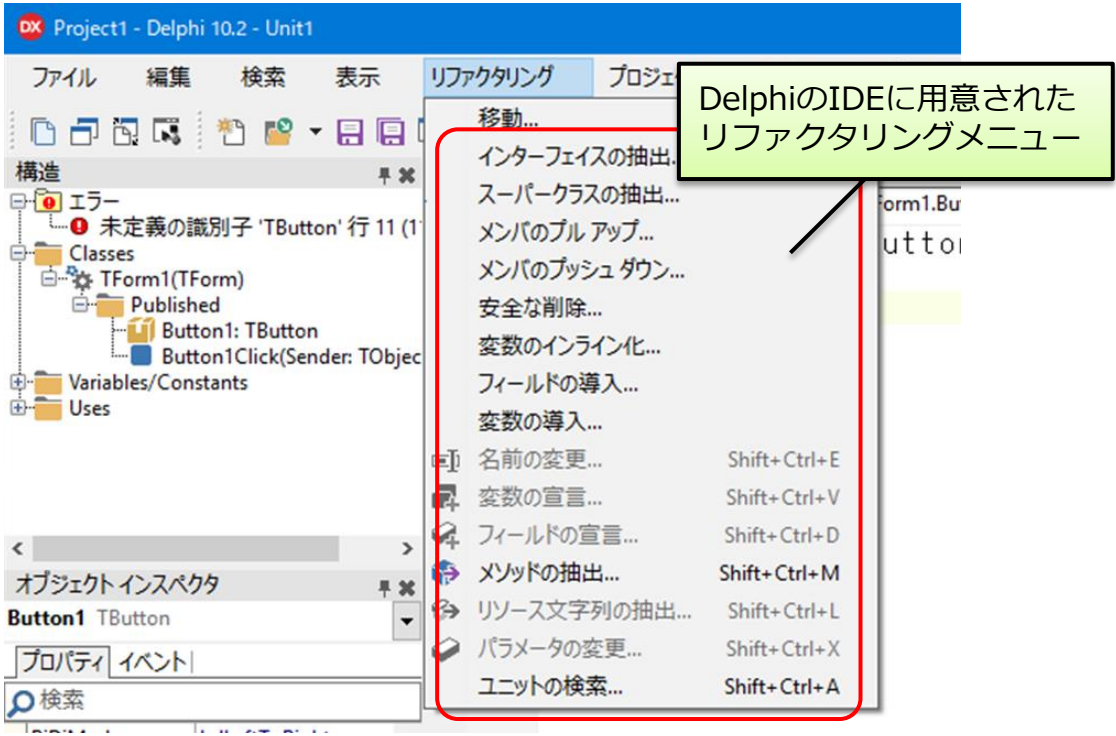
# 1-3.リファクタリング機能

## リファクタリング機能

- リファクタリングとは、プログラムの動作に影響を与えずに
  ソース、コンポーネント、クラスなどの設計を変更・整理すること。
  こうした変更をIDEの機能として適用してくれる。

＜役立つ機能＞

①名前の変更
②変数の宣言
③フィールドの宣言
④メソッドの抽出
⑤パラメータの変更



# 1-3.リファクタリング機能

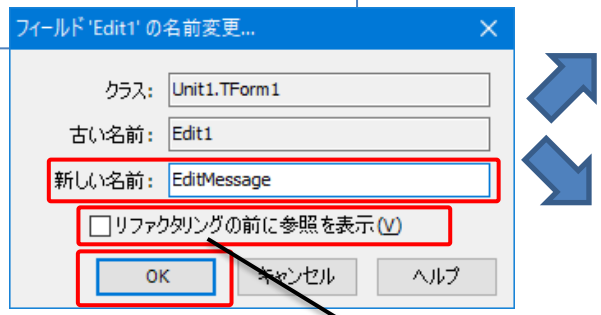
## ①名前の変更　Shift+Ctrl+E

- 一度つけた変数やコンポーネントの名前を別の名前に一括変更できる。

```
type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private 宣言 }
  public
    { Public 宣言 }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
begin
  ShowMessage(Edit1.Text);
end;
```

変数やコンポーネントを選択して、[リファクタリング]→[名前の変更]を選択。

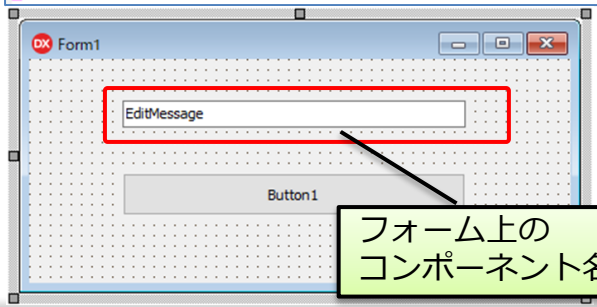
Shift+Ctrl+E でもOK




新しい名前を指定して、[OK]をクリック。

```
type
  TForm1 = class(TForm)
    EditMessage: TEdit;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private 宣言 }
  public
    { Public 宣言 }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
begin
  ShowMessage(EditMessage.Text);
end;
```

Edit1の部分が全てEditMessageに変更。




フォーム上のコンポーネント名も変更。

# 1-3.リファクタリング機能

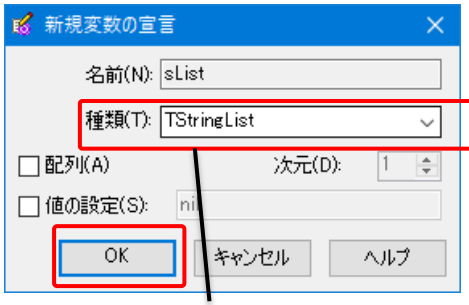
## ②変数の宣言　Shift+Ctrl+V

- コード記述中に新しい変数が出てきたとき、変数宣言部に戻らずに、自動的に宣言を追加できる。

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  sList := TStringList.Create;

end;
```

新しく追加した変数を選択して、[リファクタリング]→[変数の宣言]を選択。

Shift+Ctrl+V でもOK




データ型を指定する。
コードよりデータ型が類推される場合は候補が初期セットされる。

宣言部が自動的に追加される。

```
procedure TForm1.Button1Click(Sender: TObject);
var
  sList: TStringList;
begin
  sList := TStringList.Create;

end;
```

# 1-3.リファクタリング機能

## ③フィールドの宣言　Shift+Ctrl+D

- コード記述中に新しいフィールド（グローバル変数）が出てきたとき、Interface部に移動せずに、自動的にフィールド宣言を追加できる。

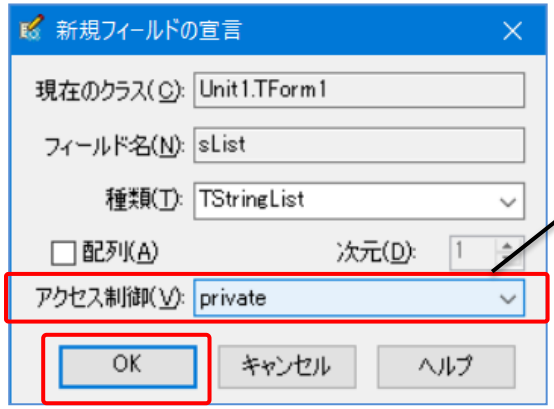
```
procedure TForm1.Button1Click(Sender: TObject);
begin
  sList := TStringList.Create;

end;
```

新しく追加した変数を選択して、[リファクタリング]→[フィールドの宣言]を選択。

Shift+Ctrl+D でもOK

```
type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    sList: TStringList;
    { Private 宣言 }
  public
    { Public 宣言 }
  end;
```

指定したアクセス制御の位置にフィールドとして追加される。




Private部、Public部等
追加したい位置を指定。

# 1-3.リファクタリング機能

## ④メソッドの抽出　Shift+Ctrl+M

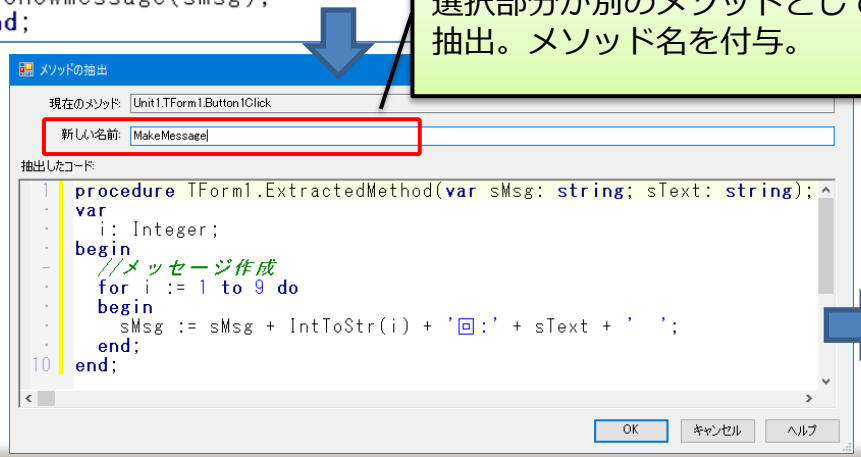
- 長い処理や同じ処理がある場合、その部分のロジックを抜き出して別のメソッドを作成することができる。

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i: Integer;
  sMsg: String;
  sText: String;
begin
  //初期化
  sMsg := '';
  sText := Edit1.Text;

  //メッセージ作成
  for i := 1 to 9 do
  begin
    sMsg := sMsg + IntToStr(i) + '回:' + sText + '  ';
  end;

  //結果の表示
  ShowMessage(sMsg);
end;
```

処理を分割したい部分を選択し、
[リファクタリング]→[メソッドの抽出]
を選択。

Shift+Ctrl+M でもOK

選択部分が別のメソッドとして抽出。メソッド名を付与。




```
procedure TForm1.Button1Click(Sender: TObject);
var
  sMsg: String;
  sText: String;
begin
  //初期化
  sMsg := '';
  sText := Edit1.Text;

  MakeMessage(sMsg, sText);

  //結果の表示
  ShowMessage(sMsg);
end;
```

元のソース部分は、メソッド呼出処理に書き換わる。

処理が分割されて、新しいメソッドが作成される。

```
procedure TForm1.MakeMessage(var sMsg: string; sText: string);
var
  i: Integer;
begin
  //メッセージ作成
  for i := 1 to 9 do
  begin
    sMsg := sMsg + IntToStr(i) + '回:' + sText + '  ';
  end;
end;
```

# 1-3.リファクタリング機能

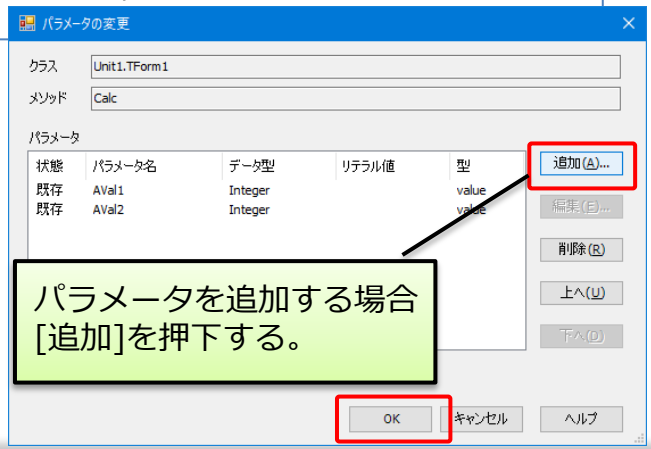
## ⑤パラメータの変更　Shift+Ctrl+X

- 定義済みのメソッドのパラメータの数や属性を変える事ができる。

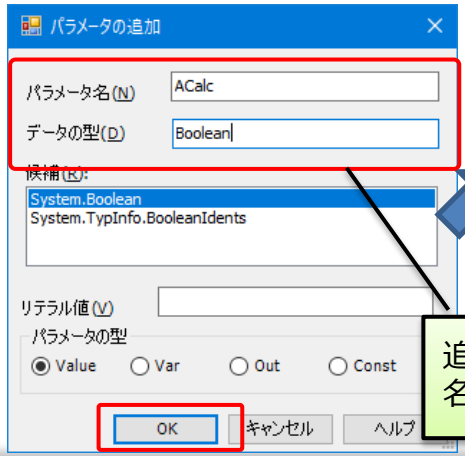
```
procedure TForm1.Button1Click(Sender: TObject);
var
  iVal1, iVal2, iRet: Integer;
begin
  iVal1 := StrToInt(Edit1.Text);
  iVal2 := StrToInt(Edit2.Text);

  //計算を行う
  iRet := Calc(iVal1, iVal2);

  Edit3.Text := IntToStr(iRet);
end;

function TForm1.Calc(AVal1, AVal2: Integer): Integer;
begin
  Result := AVal1 + AVal2;
end;
```

パラメータを変更したいメソッドを選択し、[リファクタリング]→[パラメータの変更]を選択。

Shift+Ctrl+X でもOK




パラメータを追加する場合[追加]を押下する。

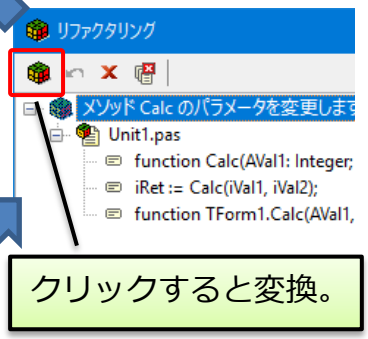



追加するパラメータの名前とデータ型を指定。

```
procedure TForm1.Button1Click(Sender: TObject);
var
  iVal1, iVal2, iRet: Integer;
begin
  iVal1 := StrToInt(Edit1.Text);
  iVal2 := StrToInt(Edit2.Text);

  //計算を行う
  iRet := Calc(iVal1, iVal2, ACalc);

  Edit3.Text := IntToStr(iRet);
end;

function TForm1.Calc(AVal1, AVal2: Integer;
  ACalc: Boolean): Integer;
begin
  Result := AVal1 + AVal2;
end;
```

メソッド呼出部分もパラメータが変更される。

パラメータが追加




クリックすると変換。

# 1-4.ソース整形機能

## ソース整形機能

- ソースを一定のコードルールにもとづいて、自動整形する機能。
  チーム開発で個人ごとにコードの記述方法やインデントの桁などの違いがある場合でも、手間をかけずに統一することができる。

```
procedure TForm2.Button1Click(Sender: TObject);
var i,j,k:Integer; Ret:Integer;
begin
  Ret:=0;
  for i := 1 to 9 do
  begin
  for j := 1 to 9 do
  begin
  for k := 1 to 9 do
  Ret := Ret+i+j+k;
  end;
  end;
```

整形実行

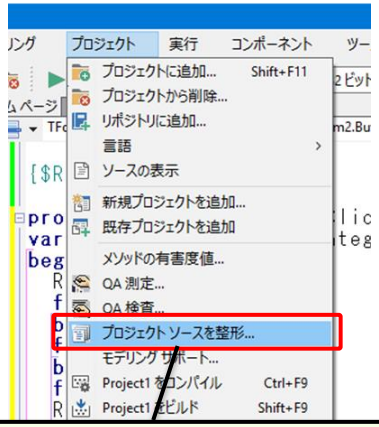
```
procedure TForm2.Button1Click(Sender: TObject);
var
  i, j, k: Integer;
  Ret: Integer;
begin
  Ret := 0;
  for i := 1 to 9 do
  begin
    for j := 1 to 9 do
    begin
      for k := 1 to 9 do
        Ret := Ret + i + j + k;
    end;
  end;
```

変数宣言部が文毎に改行される。

インデント位置が調整される。

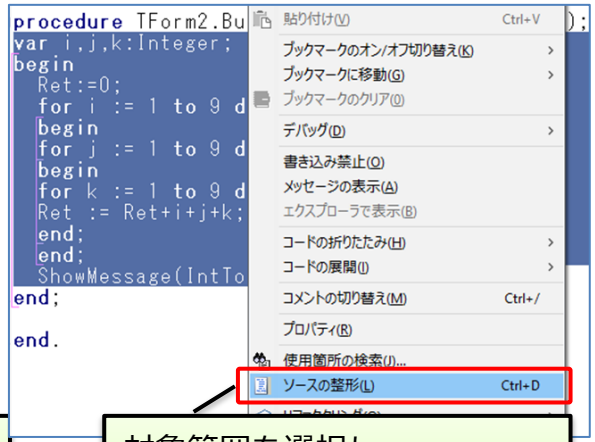
### <ソース整形の使用方法>

プロジェクト全体の整形




[プロジェクトソースを整形]を選択する。

指定した範囲のソースを整形



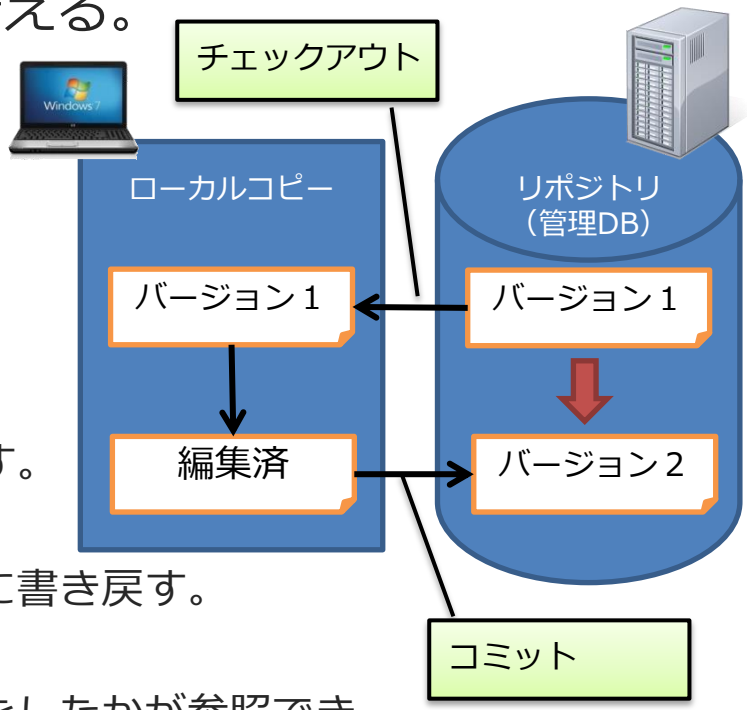

対象範囲を選択し、右クリック→[ソースの整形]を選択する。

整形のルールは、
[ツール]→[オプション]→[フォーマッタ]で設定可能

# ２．ソース管理機能

# 2-1.ソース管理ツール

- ソース管理ツールとは？
  - ソースファイルの作成・更新日時/変更内容の履歴を管理するツール。
    過去の内容や変更差異を確認したり、変更前に復元する事が可能。
    特に複数人でソースを開発する場合には、修正するソースの競合防止や上書きミスのリカバリなどの管理が行える。




- ソース管理ツールの基本機能

①インポート

ファイルをリポジトリ（管理DB）に登録する。

②チェックアウト

ファイルをリポジトリからローカル環境に取り出す。

③コミット

ローカル環境で、変更したファイルをリポジトリに書き戻す。

④変更履歴

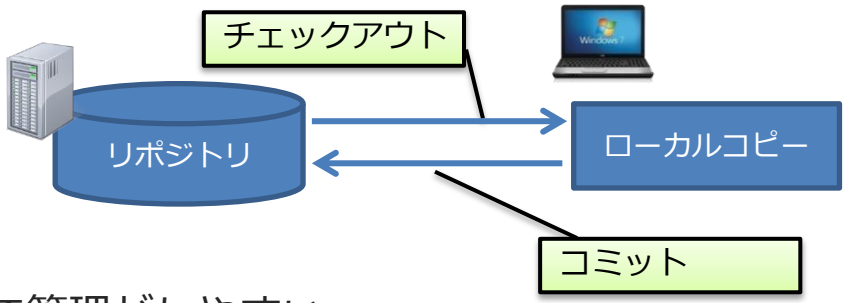
ソースのバージョン毎にいつ、誰が、どんな変更をしたかが参照でき、バージョンを戻すこともできる。

# 2-1.ソース管理ツール

- Delphi/400が標準で連携できるソース管理ツール

**Delphi/400では、オープンソース（無償）のSubversionとGitに対応したI/Fが用意されており、IDEから直接連携することができる。**

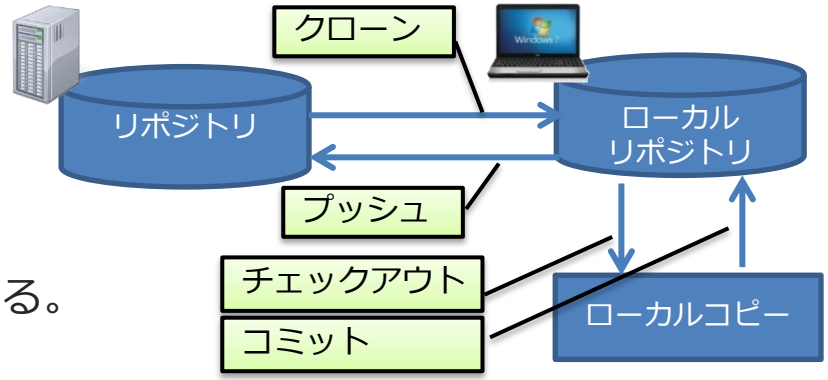
- **Subversion**（サブバージョン）
  - **集中管理リポジトリ**
    - 開発の区切り毎に単一のリポジトリにコミットを行う。




**【メリット】**

・単一リポジトリの為、シンプルで管理がしやすい。

- **Git**（ギット）
  - **分散管理リポジトリ**
    - ローカルにもリポジトリを保持し、ローカルリポジトリに対してコミットを行い、別途リモートリポジトリと同期をとる。




**【メリット】**

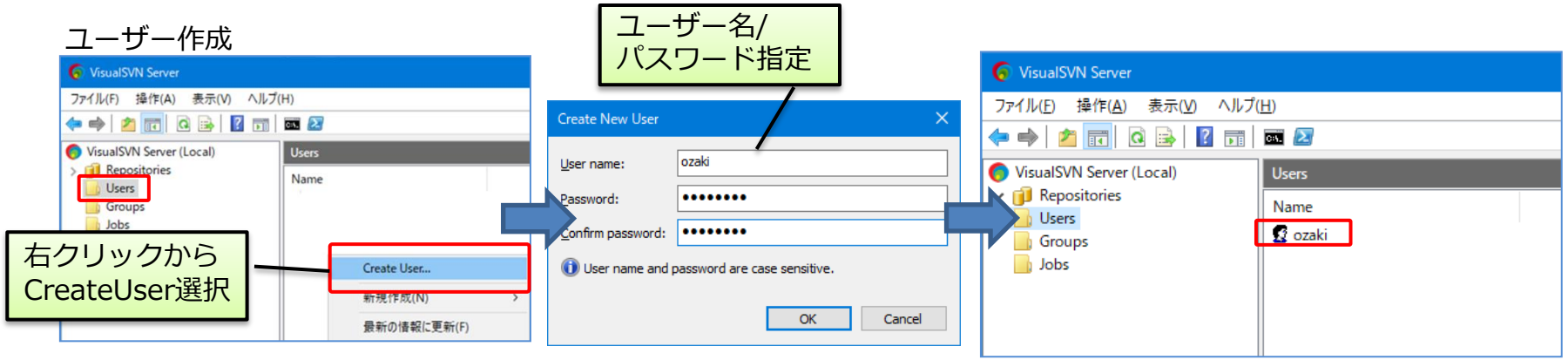
・ローカル（個人）とリモート（共通）でリポジトリが分かれているため、共通ソースに影響を与えず、個人でテスト的な変更や検証が行える。

# 2-2.環境構築(Subversion)

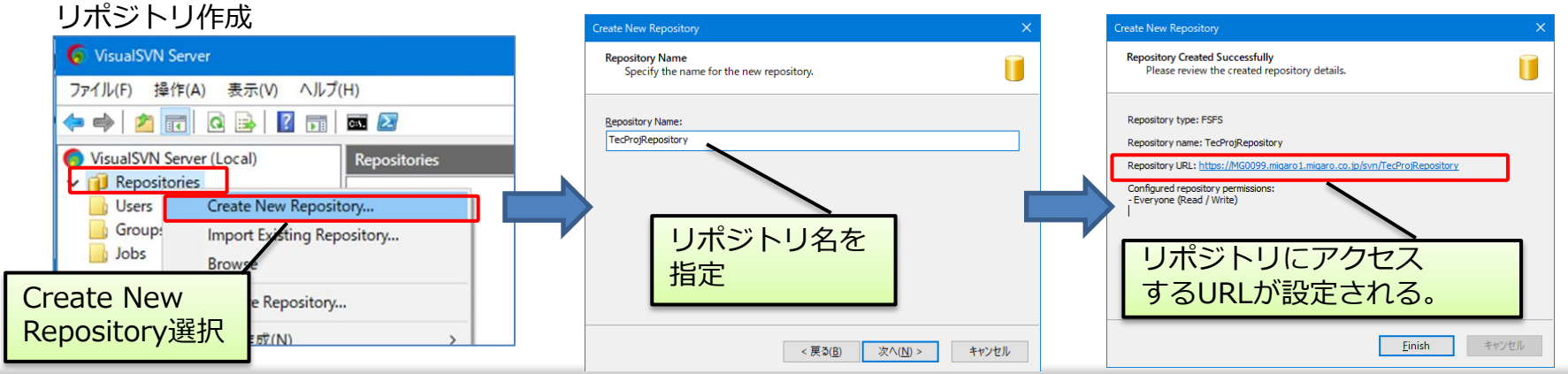
- Subversion の環境構築
  - VisualSVN Server　（GUIベースのSubversionサーバー）

    https://www.visualsvn.com/downloads/
    からダウンロードしてインストール

ユーザー作成




リポジトリ作成



# 2-2.環境構築(Git)

- Git の環境構築
  - Gitをインストール

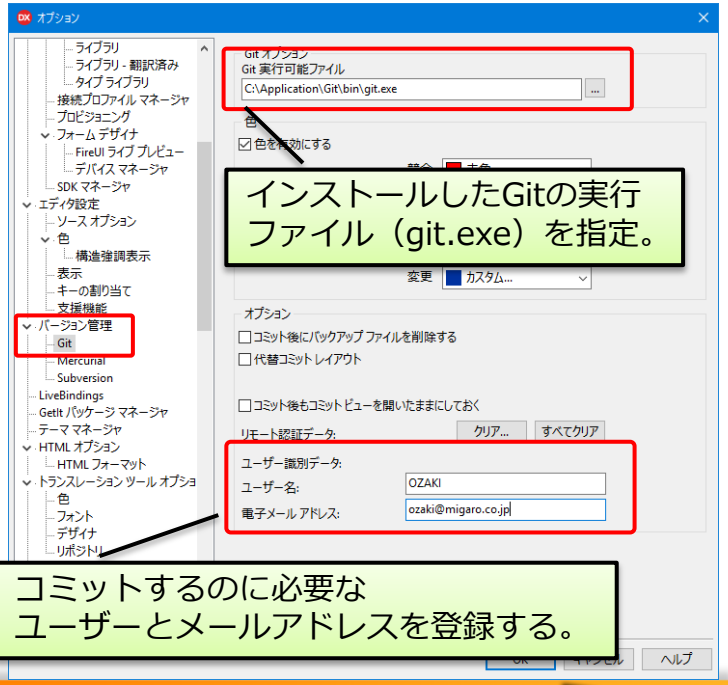
    https://git-scm.com/ よりGit for Windowsをダウンロードして
    インストール

    Gitサーバー設定（ユーザー、リポジトリ作成等）はコマンドで行う。
    （参考サイト）【Windows10パソコン上にGitサーバを立ててみた】
    http://imamachi-n.hatenablog.com/entry/2018/06/09/170331

  - DelphiでGitを使用する設定を行う。

    [ツール]→[オプション]より、
    [バージョン管理]→[Git]を選択し、

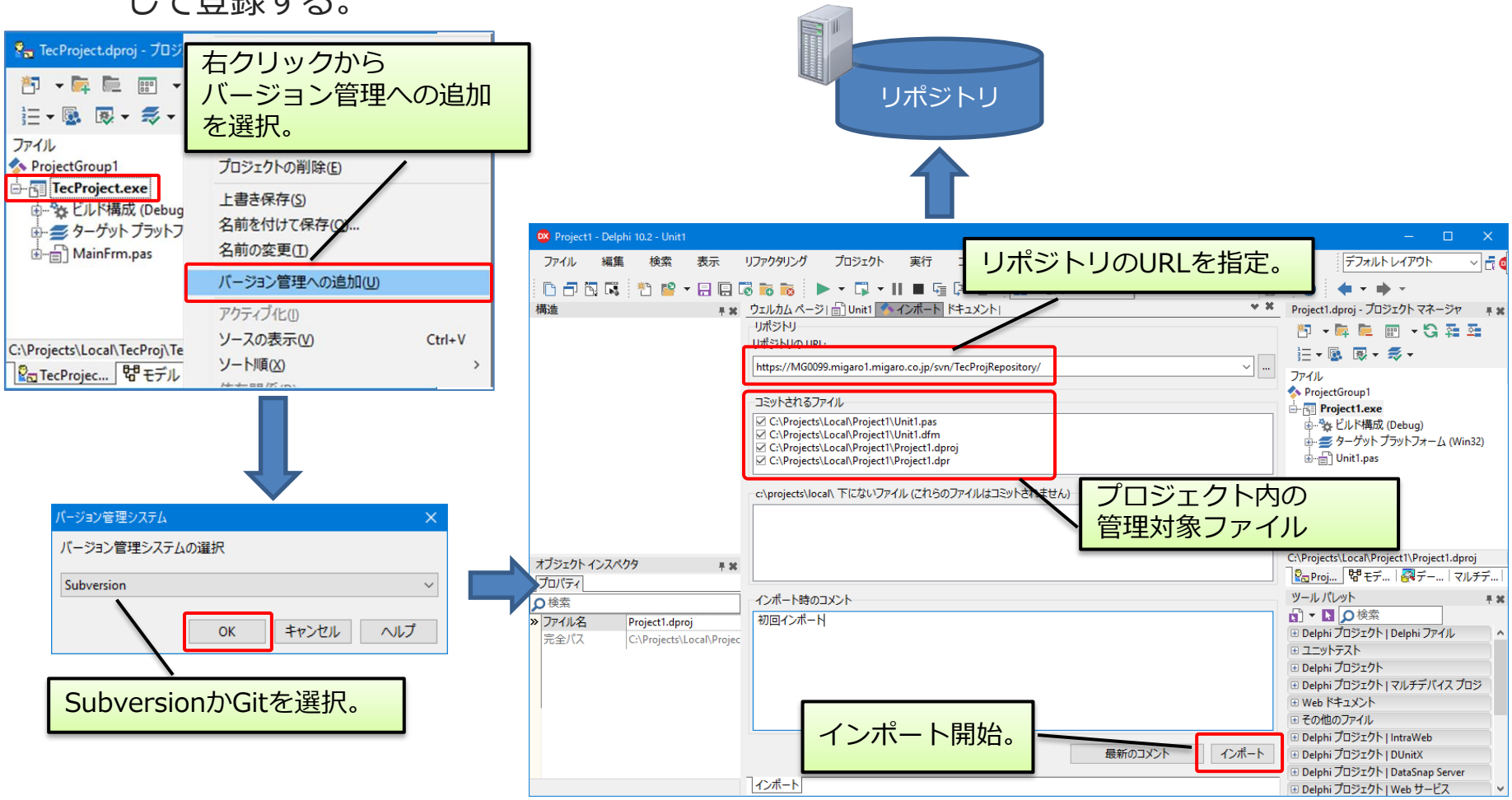
    ①git.exeの実行パスを指定する。
    ②コミットする為のユーザーと
    　メールアドレスを登録する。



# 2-3. 使用方法

## ①リポジトリへインポート（Subversion / Git共通）

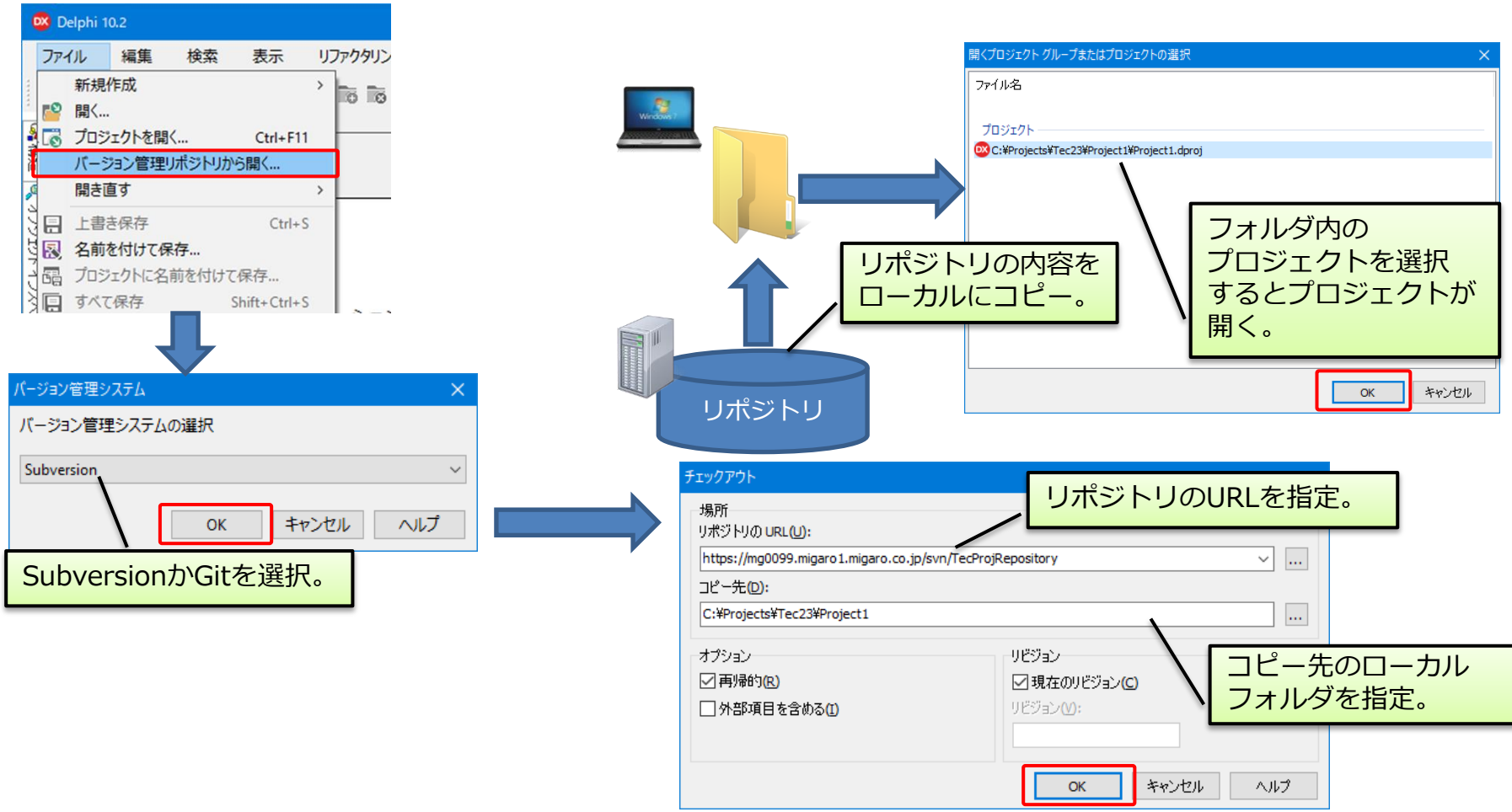
- DelphiプロジェクトをSubversionあるいはGitのリポジトリサーバーへアップロードして登録する。



# 2-3. 使用方法

## ②リポジトリからチェックアウト（Subversion / Git共通）

- リポジトリよりソースをローカルに取得する。

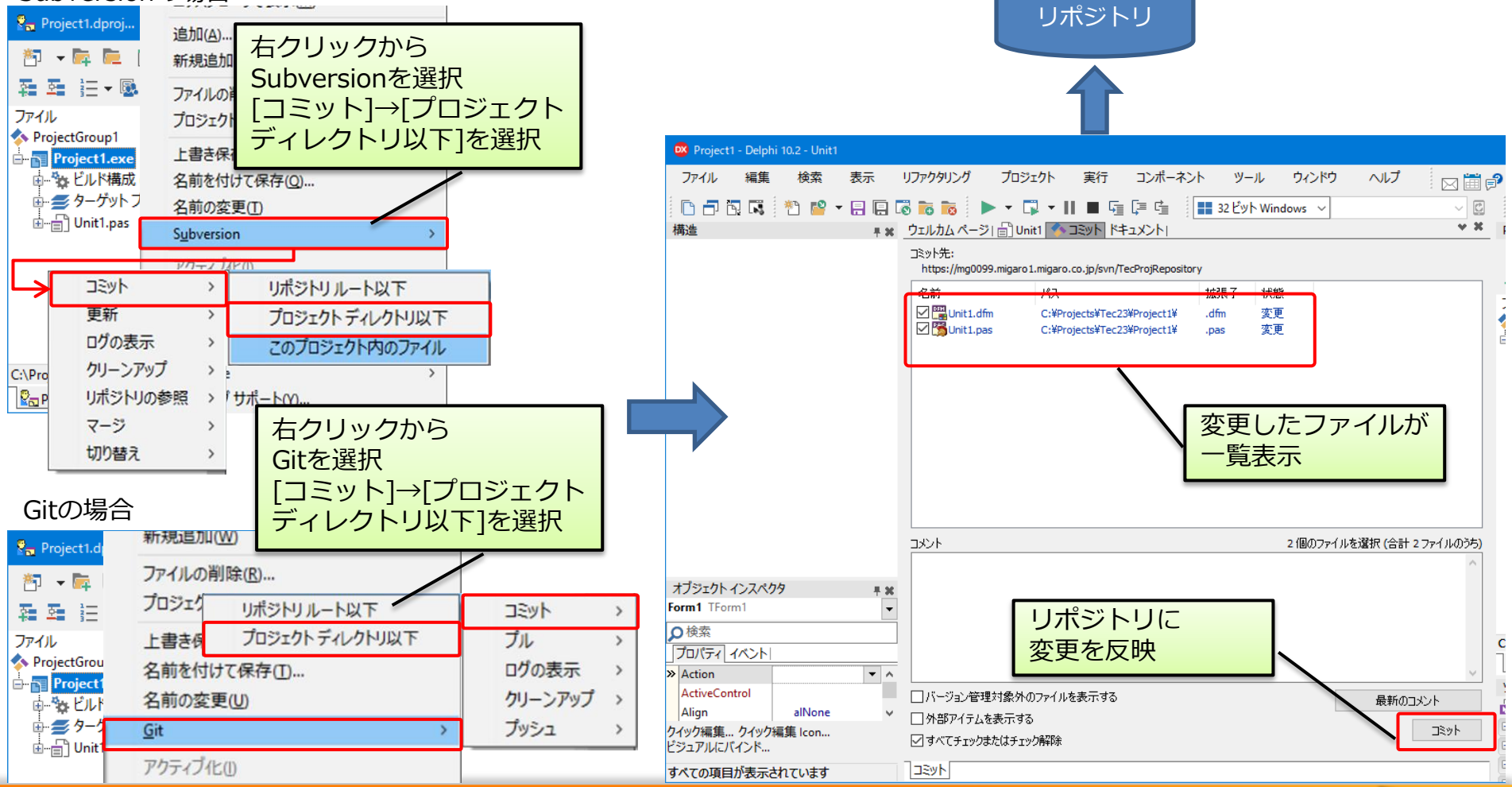


# 2-3. 使用方法

## ③リポジトリへのコミット（Subversion と Gitで異なる）

- 変更したプログラムソースをリポジトリに更新する。

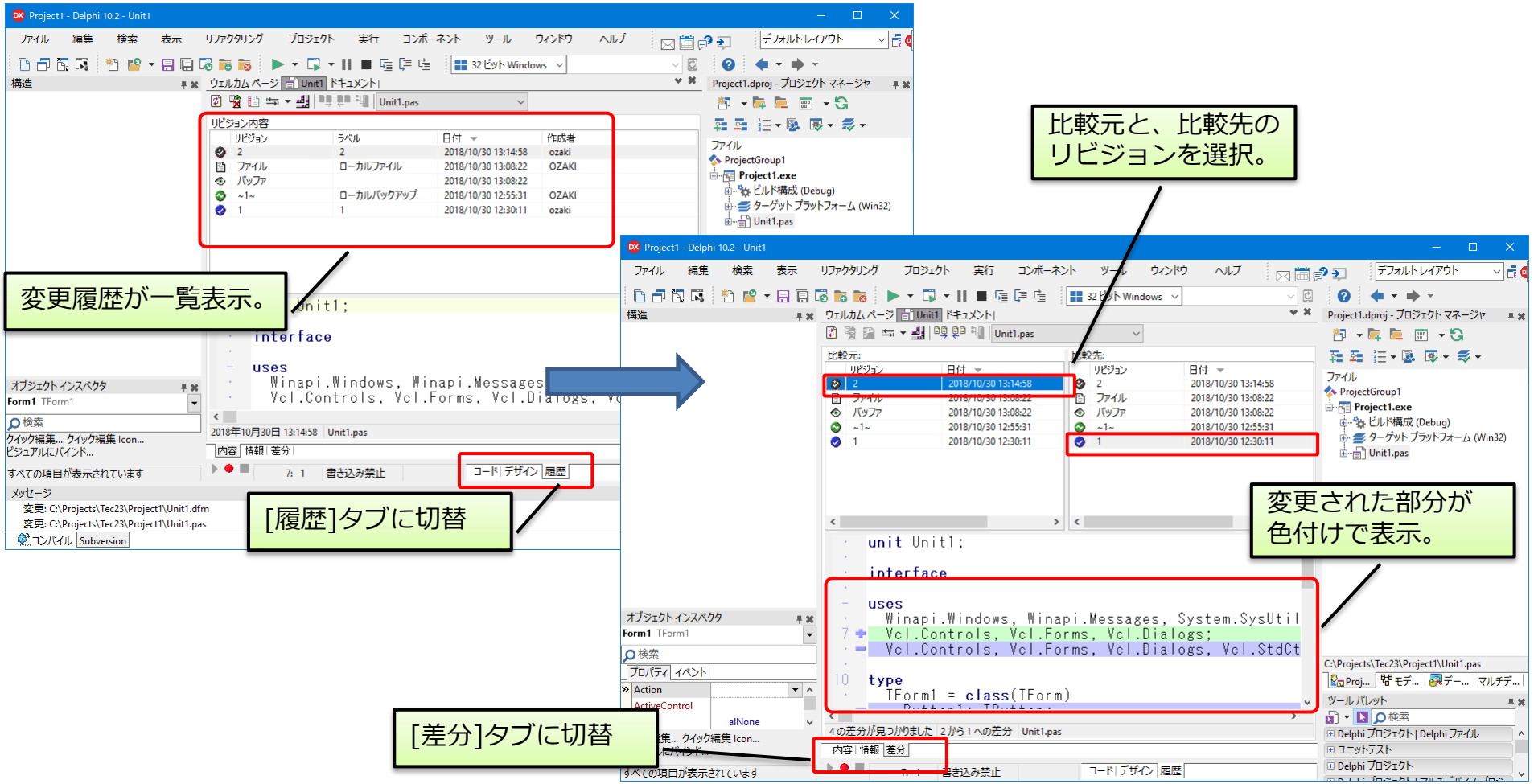
Subversionの場合



# 2-3. 使用方法

## ④変更履歴の確認（Subversion / Git共通）

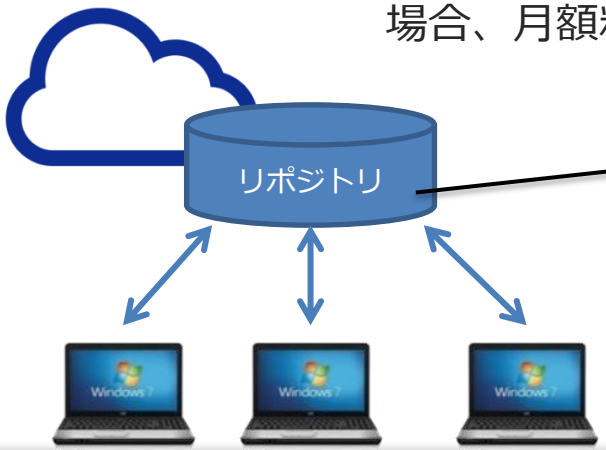
- これまでの修正履歴を選択して差分を確認する。



# 補足：GitHubの活用

- GitHub
  - Gitには、Subversion同様独自のサーバーを立てて管理する方法もあるが、GitHub（ギットハブ）を使用する方法もある。

  - GitHubとは？
    - Gitを利用したプログラムのバージョン管理が行えるウェブサービス
      - GitHubがリモートリポジトリを提供。
      - オープンソースのプロジェクトも多数公開。
      - 公開プロジェクトであれば、無料利用可能。
      - 社内アプリ等限定プロジェクトを管理する場合、月額料金制となる。

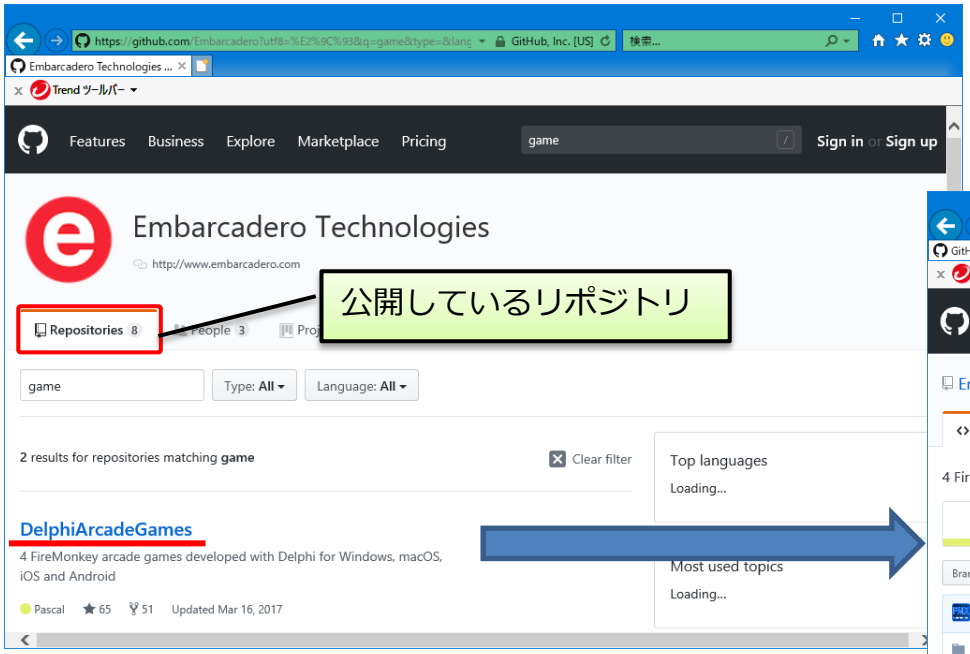


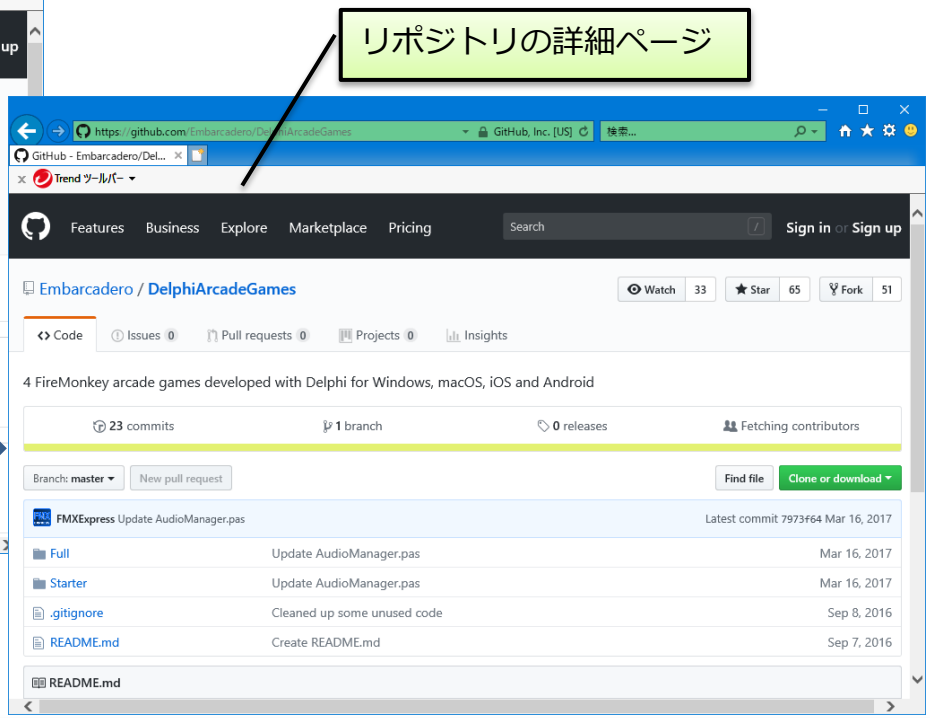

インターネット上にある
リモートリポジトリを使用する


https://github.co.jpから
アカウントを登録して利用する

# 補足：GitHubの活用

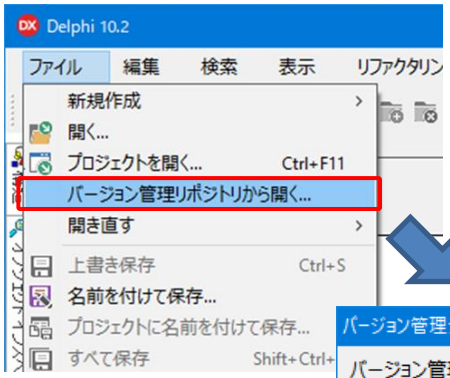
- GitHub上には、Delphiサンプルも公開されている
  - コンポーネントをはじめ各種Delphiプロジェクトを取得可能。
  - エンバカデロもGitHubアカウントを公開している。
    https://github.com/Embarcadero

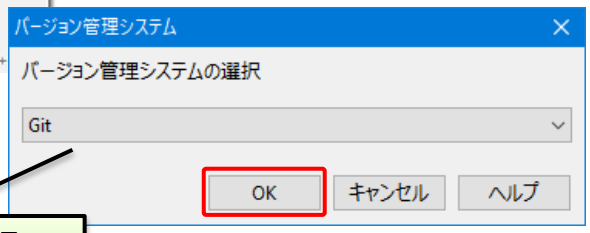


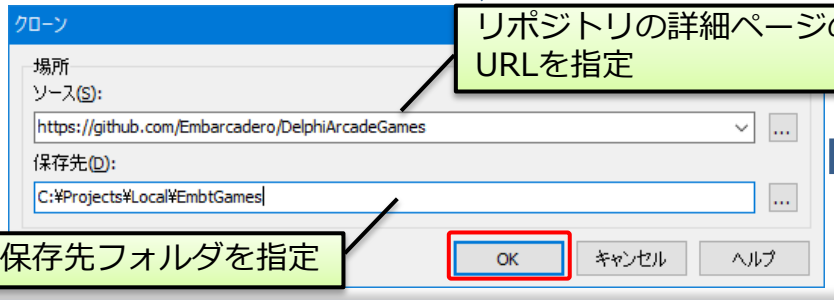
# 補足：GitHubの活用

- GitHubプロジェクトの取得
  - 公開リポジトリのプロジェクトを
    ダウンロードする。



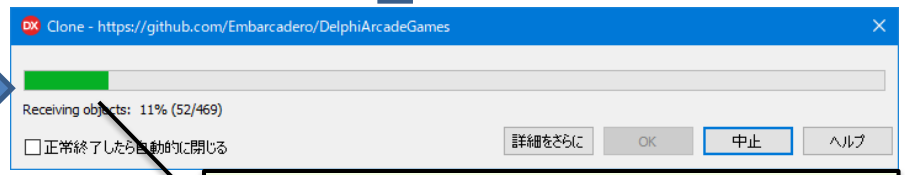




Gitを選択

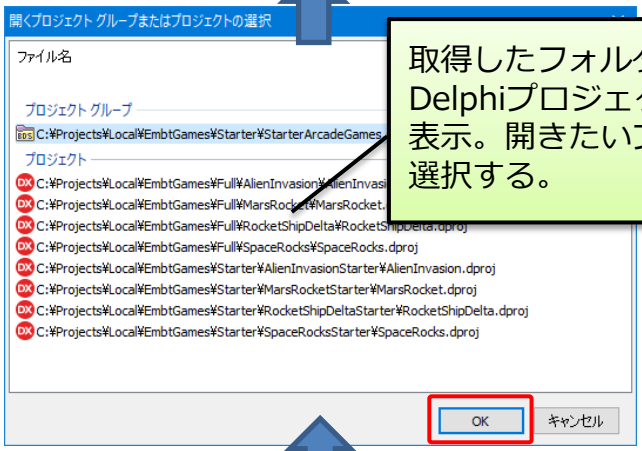



リポジトリの詳細ページのURLを指定

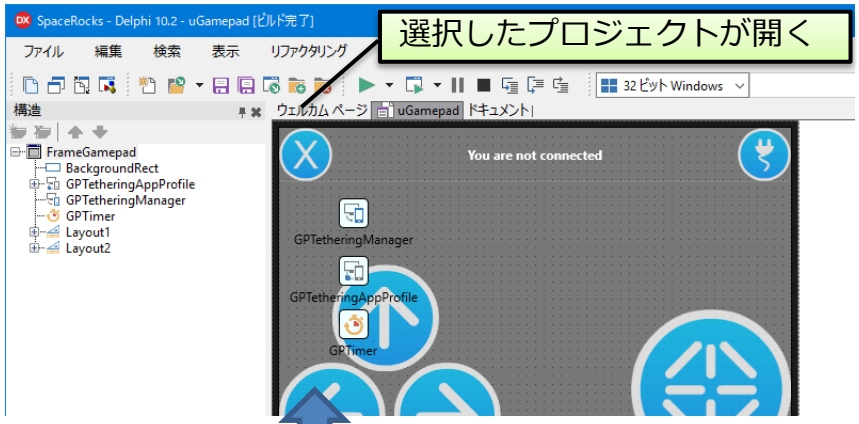
保存先フォルダを指定




リポジトリよりプロジェクトをダウンロード




取得したフォルダ内のDelphiプロジェクトが一覧表示。開きたいプロジェクトを選択する。




選択したプロジェクトが開く

# ３．まとめ

# ■ まとめ

- ソース開発機能
  - コード支援、ライブテンプレート機能を使用することで、コーディング作業を効率的に行える。
  - ソースの変更や整理はIDEのリファクタリング機能で安全に行える。
  - ソース整形機能を使えば、チーム内のコーディングを統一できる。

- ソース管理機能
  - Delphi/400のIDEには、Subversion及びGitに対応するクライアント機能をもっており、IDE上から連携できる。
  - GitHubプロジェクトは、オンライン上で公開されているサンプルのダウンロードをすることができる。