Object Pascal "超" 強い静的型付け言語の魅力

ODC 2022 - D会場
Learn Languages 2022

2022/09/03 初版 細川 淳

S/G シリアルゲームズ
HAPPY CREATION, PLAY THE DEVELOPMENT!

アジェンダ

- 今回は型を切り口に Object Pascal を紹介します。

- **自己紹介**
- **最近の Delphi の紹介**

- **Pascal の紹介**
  - 概要・歴史
  - 型の紹介

- **Object Pascal の紹介**
  - 概要・歴史
  - 型の紹介

- **動的な型の導入**

自己紹介

- **細川 淳**
  - 株式会社シリアルゲームズ 取締役
  - Embarcadero MVP (Delphi MVP)
  - Twitter: @pik

  - **技術領域**
    - Windows / macOS / iOS / Android
    - 画像処理
    - 通信処理
    - UI / UX 研究

  - **好きな言語**
    - Object Pascal
    - C#
    - Dart

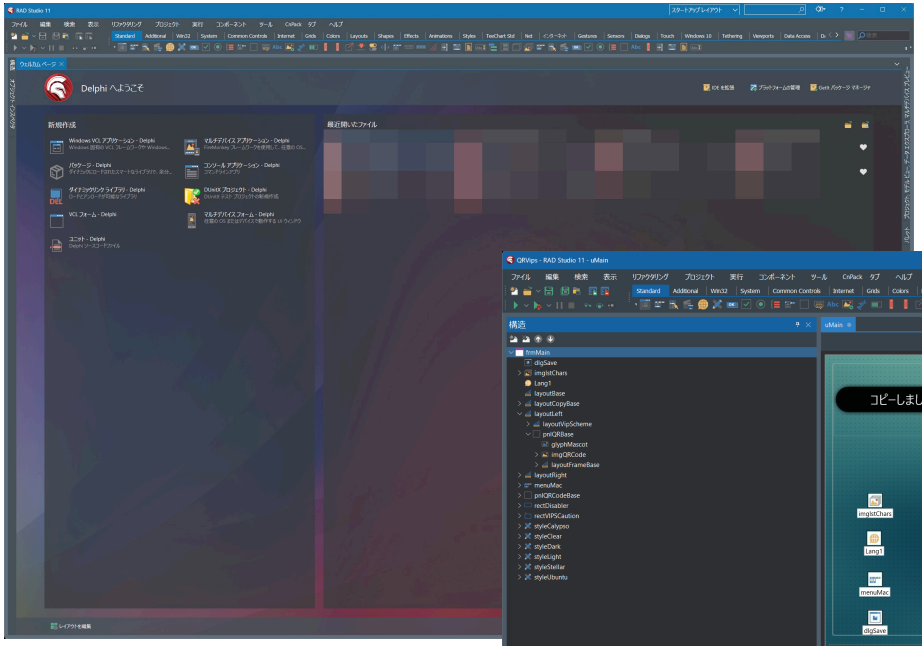
東京ゲームショウ2022
ビジネス ソリューション コーナー
4-N33

シリアルゲームズ公式キャラクター
シリア

最近の Delphi の紹介

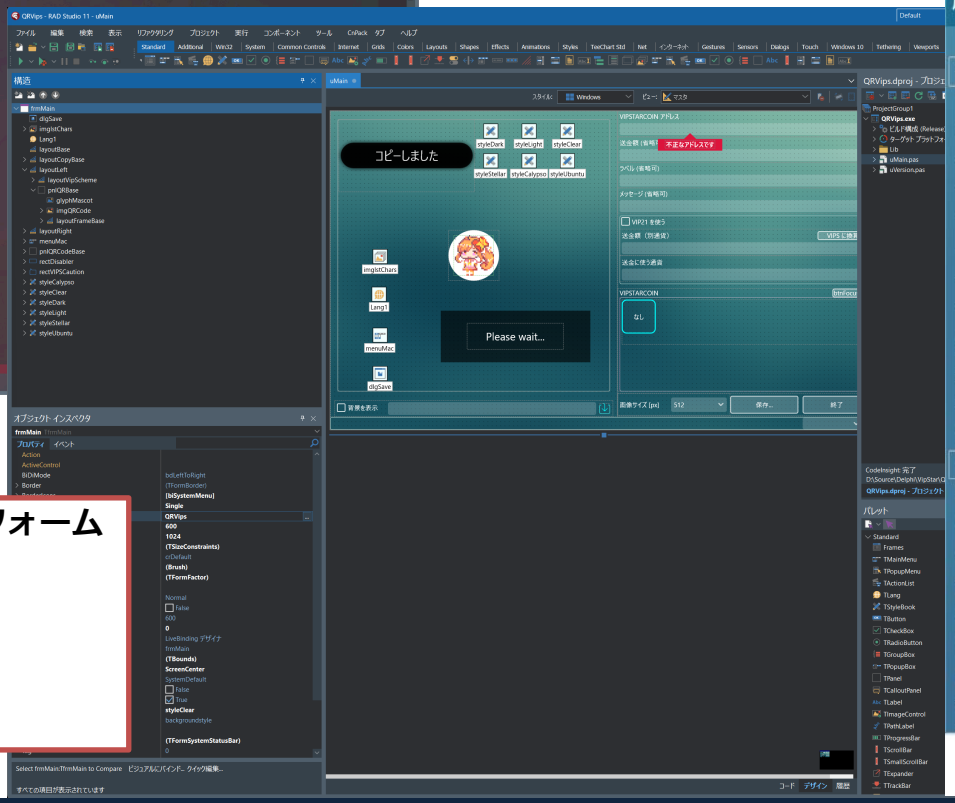
最近の Delphi

- 最近の Delphi
  - 開発販売はエンバカデロ・テクノロジーズ（Embarcadero Technologies）



ウェルカムページ

設計画面



Delphi で作ったモバイルアプリ

Delphi で開発できるプラットフォーム

- Windows
- macOS
- Linux
- iOS
- Android

最近の Delphi

- 最近の Object Pascal を使える唯一と言って良い環境
  - FreePascal など他にも Object Pascal を使える言語はありますが Delphi の後追いになります
- 無料の Community Edition があります
  - ライセンス違反が多かったため現在 Community Edition は 1 つ前のバージョン 10.4 が提供されています
- 興味を持ったら使って見てください

https://www.embarcadero.com/jp/products/delphi/starter

Pascal 編

- **注意事項**
  - 講演者は「純粋な Pascal」を触ったことがないため「Turbo Pascal」基準で書いています。
  - 純粋な Pascal や Pascal の系譜については @ht_deko さんの Qiita の記事が詳しいです
    - このスライドの作成にも多くのアドバイスをいただきましたありがとうございます

Twitter: @ht_deko

Pascal へのオブジェクト指向拡張の歴史と Delphi
https://qiita.com/ht_deko/items/ff2d90c9174290583377

標準 Pascal 範囲内での Delphi 入門
https://qiita.com/ht_deko/items/e2796788c65c2cda1de5

- **Pascal の特徴**
  - 1970 年誕生 (52歳！)
  - ニクラウス・ヴィルト (Niklaus Wirth) 博士が制作
    - Pascal を初めとしてプログラミング言語（Modula-2 など）を多数制作
      名著「アルゴリズム＋データ構造＝プログラム」等があります
  - 特徴
    - ALGOL をベースに言語仕様を簡素化 / LL(1) 文法
    - 大文字と小文字を区別しません
    - 予約語と指令
      - 指令はコンテキストによって予約語相当に
    - 戻り値は関数名に値を代入します
      - Pascal では「戻り値」ではなく「結果」と呼びます
      - 再帰がやりづらいため Turbo Pascal では
        Result 変数が導入されています
  - 教育目的
    - Pascal で Pascal を実装できます
      ⇒ Bootstrapping
    - **非常に強い静的型け言語**

- **型 (Turbo Pascal 準拠)**
  - 単純型
    - **順序型**
    - 実数型
  - 文字列型
    - 特殊な Char の配列
      - 255 文字まで格納可能
  - 構造化型
    - record 型
      - 共用体も record で定義できる
  - ポインタ型
  - **集合型**
  - ファイル型
    - Delphi になってからは使われることはほぼ無い

```
// FizzBuzz
program FizzBuzz(out);
type
  TRange = 1.. 100; // 部分範囲型
var
  I: TRange;
  S: String;
begin
  for I := Low(TRange) to High(TRange) do
  begin
    S := '';

    if I mod 3 = 0 then
      S := 'Fizz';

    if I mod 5 = 0 then
      S := S + 'Buzz'; // 標準 Pascal では不可

    if S = '' then
      Str(I, S);

    WriteLn(S);
  end;
end.
```

- **順序型**
  - 順序のある値
    - 比較演算 (<, >, = など) 可能
    - 特殊な定義済み関数を使えます
      - Ord　　順序値
      - Pred / Succ　　前・次の値
      - Low / High　　最小値・最大値
  - Pascal における順序型
    - 整数型
    - 文字型
      - Turbo Pascal では
        0～255 の文字コード(処理系依存)
    - 論理型（！）
      - False.. True
    - 列挙型
      - ユーザー定義の列挙値
    - 部分範囲型
      - 既存の順序型の部分範囲

```
program TypeSample(out);
type
  TRange = 1.. 100;       // 部分範囲型
  TEnum = (Cat, Dog, Fox); // 列挙型
var
  I: Integer; // 整数型
  C: Char;    // 文字型
  B: Boolean; // 論理型
  E: (Nyaa, Wan, Kon); // 列挙型
begin
  for C := 'A' to 'Z' do
    Writeln(C); // 大文字を出力

  for C := 'A' to 'Z' do
    Writeln(Ord(C)); // 文字コードを出力

  for C := 'A' to 'Z' do
    if C > 'G' then  // G より上のみ出力
      Writeln(C);
end.
```

- **集合型**
  - Pascal はライブラリではなく言語の機能として「集合」を扱えます。
    - 値は順序型
    - Turbo Pascal での要素数は 256 個まで
      - 要素数は処理系依存
  - 集合の演算

意味	数学	Pascal	例
相当	=	=	if A = B then ...
不一致	≠	<>	if A <> B then ...
部分集合	⊂	<=	if A <= B then ...
上位集合	⊃	>=	if A >= B then ...
所属	∈	in	if A in X then ...
和集合	∪	+	if A + B <> [] then ...
差集合	＼	-	if A - B = [] then ...
積集合	∩	*	if A * B <= X then...

```
program SetSample(out);
var
  Alpha: set of 'a'.. 'z';
  A: Char;
begin
  // [] は集合構成子
  Alpha := ['f', 'o', 'x'];

  A := 'f';
  if A in Alpha then
    Writeln('属しています'); // 表示される

  A := '0';
  if A in Alpha then
    Writeln('属しています'); // 表示されない

  // 追加
  Alpha := Alpha + ['c', 'a', 't'];

  A := 'a';
  if A in Alpha then
    Writeln('属しています'); // 表示される
end.
```

- **演算子**
  - 強い静的型付け言語のため実数と整数の演算子が異なる部分があります！
  - 除算
    - 実数 R := F / 2;
    - 整数 I := N div 2;
  - 剰余
    - 実数 未サポート
    - 整数 I := N mod 2;

```
var
  I: Integer;
  R: Real;
begin
  I := 20;
  R := 4.2;

  I := I div 2; // OK
  R := I / 2;   // OK
  R := I div 2; // OK

  I := R div 2; // NG
  I := R / 2;   // NG
end.
```

- **型の代入互換性**
  - 代入や演算できるのは同じ型のみ
    - Cast すれば代入できます

C / C++ は型が違っても代入できる

```
int main(int argc, _TCHAR* argv[])
{
  int a = 0;
  bool b = true;

  a = b; // 型が違っても代入できる
  printf("%d¥n", a);

  char c = 'a'; // c の場合 char は整数型
  c = c + b;    // 型が違っても演算できる
  printf("%c¥n", c);
}
```

Pascal は型が違うと基本的には代入できない

```
program TypeSample2(out);
var
  I: Integer;  // 整数型
  B: Boolean;  // 論理型
  R: Real;     // 実数型 (Object Pascal では非推奨)
begin
  I := 0;
  B := True;
  R := 4.5;

  I := B; // エラー
  R := I; // 実数の表現範囲の方が整数より広いため OK
end.
```

Object Pascal 編

- **Object Pascal の特徴**
  - 1990 年誕生 (Turbo Pascal 基準)
    - Apple Object Pascal
      - ラリー・テスラーのチームとニクラウス・ヴィルト博士が Object Pascal を開発 (1985)
    - Turbo Pascal
      - Turbo Pascal 5.5 で Object 指向拡張が導入されました
      - アンダース・ヘルスバーグ（Anders Hejlsberg）が開発
        - » 後に **Delphi, C#, TypeScript を開発！**
  - 単純な構造 (?)
    - C++ のような多重継承はありません
    - C++ の Template 相当の Generics や
      演算子オーバーロード、
      Java の interface 相当の機能は後から追加されました
    - 今では実は少し複雑かも…？
  - **インライン変数宣言の導入**
  - **柔軟な強い静的型付け言語**

- **柔軟な強い静的型付け言語**
  - type の拡張
    - 既存の型を新しい型として定義できます
  - RTTI (RunTime Type Information)
    - 実行時型情報があります
    - リフレクションが使える (拡張 RTTI)
  - 動的なこともできちゃう！？

```
type
  // 既存の型を新しい型として宣言できる
  TInt = type Integer;

procedure Foo(var V: TInt);
begin
  WriteLn(V);
end;

begin
  var I: Integer := 0;
  Foo(I); // エラー！
end.
```

```
uses
  System.Rtti, FMX.Types;

type
  TFoo = class
    private var FBar: Integer;
  end;

begin
  var Foo := TFoo.Create;
  try
    var RT := SharedContext.GetType(Foo.ClassType);
    var Bar := RT.GetField('FBar');
    Bar.SetValue(Foo, 1);

    WriteLn(Foo.FBar); // 1 が出力される
  finally
    Foo.Free;
  end;
end.
```

- **インライン変数宣言**
  - コンパイラバージョン 33.0 以降 (Delphi 10.3 以降)
  - Pascal は構文がブロック構造となっています
    - 変数宣言は変数宣言ブロックで宣言する仕組みでした
  - Object Pascal ではインライン変数宣言が可能
    - 厳格な型があるため右辺値から左辺値の型を推測できます（型推論）

```
// Pascal の場合
const
  // 定数ブロック
  // Pascal の定数に型は指定できない
  Quux = 5.2;
var
  // 変数ブロック
  Foo: Integer;
  Bar: Integer;
  Baz: String;
  Quux: Real;
begin
  Foo := 100;
  Bar := 200;
  Baz := 'Cat';
end.
```

```
// インライン変数宣言と型推論
begin
  // インライン変数宣言（型指定）
  var Foo: Integer := 100;

  // インライン変数宣言（型推論）
  var Bar := 200;
  var Baz := 'Cat';

  // インライン定数宣言 (型指定: 型付き定数)
  const Quux: Single = 5.2;

  // インライン定数宣言 (型推論)
  const Qux = 4.2;
end.
```

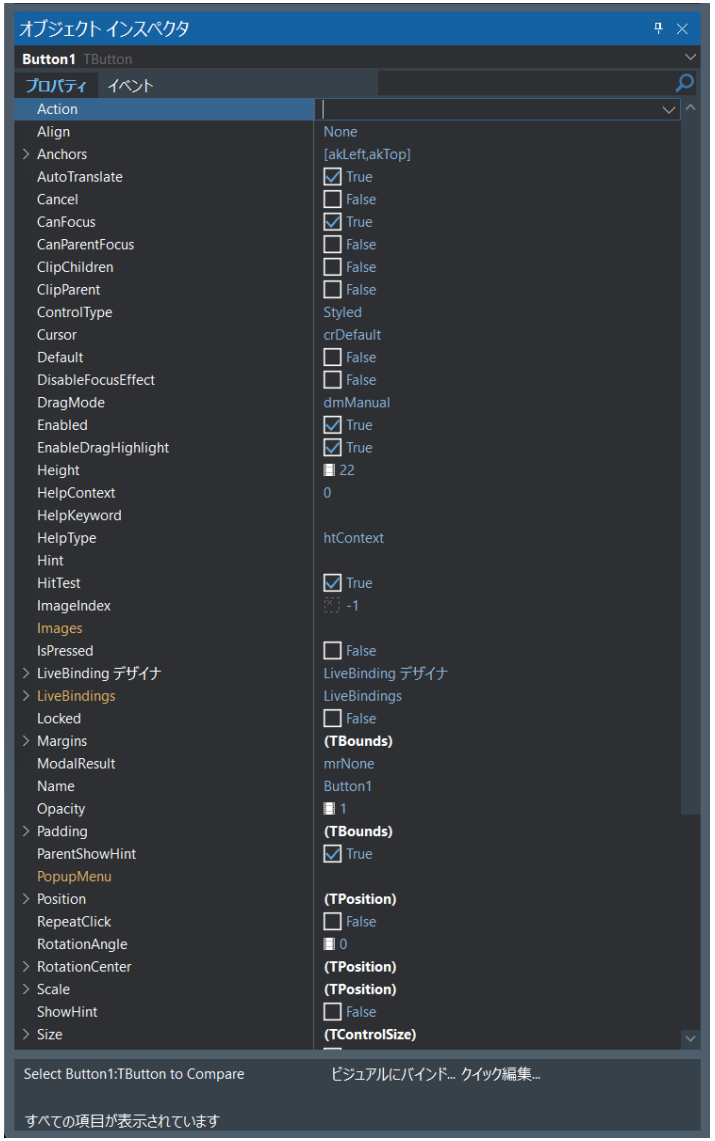
- **Object Pascal で追加された型の一部**
  (Delphi 11 Alexandria までに)
  - **クラス型(class)**
  - インターフェース型(interface)
  - **メタクラス型**
  - 高度なレコード型 (record)
  - **動的配列 (array of)**
  - **長い文字列型 (string)**
  - メソッドポインタ型
  - **無名メソッド型**
  - **Variant 型**
  - などなどなど

- **class**
  - ヒープ領域にメモリを取ります
  - property 機構の存在
  - アクセス制御

アクセス制御	範囲
strict private	自分自身
private	自分自身, 同一Unit
strict protected	自分自身, 継承先
protected	自分自身, 継承先, 同一Unit
public	誰でも
published	超誰でも！ (RTTI が生成される)

Object Pascal は C++ のような Friend はありません。
代わりに Unit Friend と呼ばれる機構があります。
同一 Unit 内では private / protected 関係なく見えちゃう！
それを防ぐのが strict 指令です。

published は RTTI が生成されるため「外」から見えるようにできます。
IDE は published 指定されたプロパティを編集できるようにしています（右図）。



- class 例

```
type
  TFoo = class
  private class var // クラス変数を宣言できます
    FBar: String;
  private var       // こちらは普通のフィールド変数 (var は省略可)
    FQuux: Integer;
  protected
    procedure Baz;
    procedure SetQuux(AValue: Integer);
    function GetQux(const AIndex: Integer): String;
    procedure SetQux(const AIndex: Integer; AValue: String);
  public
    constructor Create; reintroduce; // Object Pascal のコンストラクタは自由に名前を決められます
    destructor Destroy; override;
    property Quux: Integer read FQuux write SetQuux; // property の実体には変数やメソッドを指定できます
  published
    property Qux[const AIndex: Integer]: String read GetQux write SetQux; default;
  end;

begin
  // コンストラクタを呼び出す事でヒープにメモリを確保し初期化されます
  var Foo := TFoo.Create;
  try
    Foo[0] := 'Hello'; // default 指定された property は名前を省略できます
  finally
    Foo.Free; // 確保されたメモリを解放
  end;
end.
```

この形は良く使うので
try / ctrl + j
で一発で呼び出せます
(Live Template)

- **メタクラス**
  - クラスを表す型
  - 多態性を実現する方法の１つ
    - ファクトリで生成するインスタンスの型を動的に変更するなど

```
type
  TFoo = class
  public
    procedure Bar; virtual;
  end;

  TBaz = class(TFoo)
  public
    procedure Bar; override;
  end;

  // メタクラスの宣言（ただし今回は未使用）
  TFooClass = class of TFoo;

procedure TFoo.Bar;
begin
  WriteLn('TFoo');
end;

procedure TBaz.Bar;
begin
  WriteLn('TBaz');
end;
```

```
begin
  // 型推論で自動的に class of TFoo のメタクラスになります
  var FooClass := TFoo;

  var Foo := FooClass.Create;
  try
    Foo.Bar; // TFoo が出力されされます
  finally
    Foo.Free;
  end;

  // TFoo を継承したクラスなら代入可能
  FooClass := TBaz;

  Foo := FooClass.Create;
  try
    Foo.Bar; // TBaz が出力されます
  finally
    Foo.Free;
  end;
end.
```

- **無名メソッド型**
  - Pascal では考えられませんが、直接メソッドを渡せるようになりました
    - でも型に厳格なので型を定義しないと使えません
  - Pascal としては完全に異端ですが即時関数も定義できるようになりました…

```
type
  // 無名メソッドの型定義
  TSampleProc =
    reference to function(const A: Integer): Integer;

// TSampleProc 型の関数を受取る
procedure Foo(const AProc: TSampleProc);
begin
  WriteLn(AProc(2));
end;

begin
  // TSampleProc 型の関数を直接書ける
  Foo(
    function(const A: Integer): Integer
    begin
      Result := A * A;
    end
  );

  ReadLn;
end.
```

```
begin
  var B :=
    (function(A: Integer): Integer begin Result := A * 2; end)(3);

  WriteLn(B);
end.
```

- **動的配列型**
  - 通常配列は指定した要素数を増減できませんが、
    Object Pascal の動的配列は要素数を増減できます！
  - array of と書き、要素数を指定しなければ動的配列になります
    - 通常は array [0.. 9] of Integer などと要素数を指定します
  - SetLength ビルトイン関数でサイズを変更します
    - サイズが大きくなっても今までのデータは保持されます
  - 集合の様に足し算でサイズを変更する事も可能です

```
var
  Foo: array of String; // array はインライン変数宣言では定義できない
begin
  var Bar: TArray<String>; // array of T となっている Generics を使える

  var S := ['Cat', 'Dog', 'Fox']; // 型推論で S は 自動的に array of String 型になる
  WriteLn(S[0]); // Cat が出力される

  S := S + ['Bear', 'Turtle'];
  WriteLn(S[3]); // Bear が出力される

  SetLength(S, 10); // 確保されたメモリは初期化される（既存のデータがある場合は保持される）
  WriteLn(S[9]); // 空文字が出力される
end.
```

- **長い文字列型**
  - 参照カウントで管理されます
    - メモリは自動廃棄されます
    - 同じ文字列であればコピーをしないため非常に高速です（参照カウントが１つ増えます）
    - 文字列の加算も非常に高速です
  - 文字列長の最大値は 4GB
    - Pascal の String は 255文字
  - Object Pascal の文字列は多種
    - UnicodeString
      - 現在の String は UnicodeString のエイリアスです
    - AnsiString
      - かつて String のエイリアスだった文字列型です
      - 現在はコードページ指定に使えるように拡張されています
    - ShortString
      - Turbo Pascal の文字列
      - 短い文字列型と言われます
    - などなど
  - コードページ指定で文字列型を生成できる

```
type
  UTF8String = type AnsiString(65001); // UTF-8
  SJISString = type AnsiString(932);   // Shift-JIS
  EUCJString = type AnsiString(20932); // EUC-JP
```

- **文字列の自動変換**
  - 長い文字列は代入時に自動的にコードページ変換が走ります

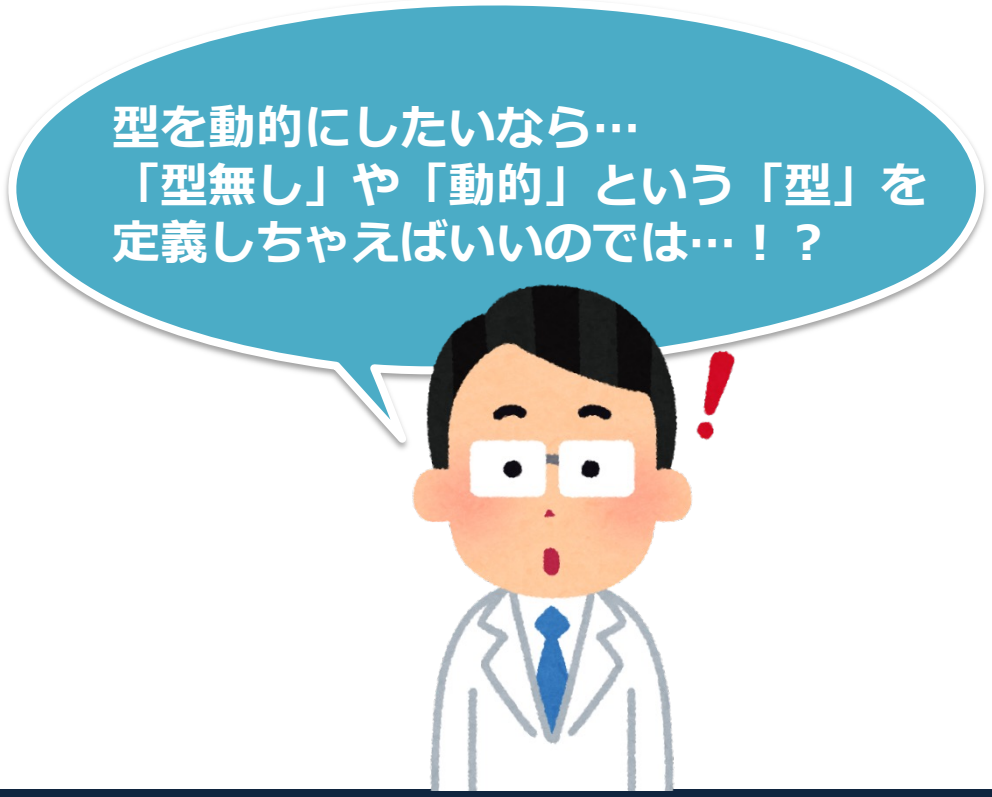
```
type
  SJISString = type AnsiString(932);   // Shift-JIS
  UTF8String = type AnsiString(65001); // UTF-8
  EUCJString = type AnsiString(20932); // EUC-JP

begin
  var S := AnsiString('あいうえお'); // AnsiString のデフォルトは Shift-JIS

  // SJIS 以外は自動的にコードページ変換が走ります
  var SJIS: SJISString := S;
  var UTF8: UTF8String := S;
  var EUCJ: EUCJString := S;

  // どれも文字化けせずに表示されます
  Writeln('SJIS: ', SJIS);
  Writeln('UTF8: ', UTF8);
  Writeln('EUCJ: ', EUCJ);
end.
```

- **文字配列の自動変換**
  - 文字の配列は String に代入できちゃう
  - Null 終端文字列も相互に変換できちゃう…
    - C/C++ のライブラリを呼び出すのが非常に簡単

```
uses
  Winapi.Windows;

procedure Sample;
var
  C: array [0.. 2] of Char;
begin
  C := 'Cat'; // 長さが合っている文字列は代入可能

  // 静的配列は文字列に変換可能
  var S: String := C;
  Writeln(S);

  // PChar として型変換すると Null 終端が自動的に付く…
  MessageBox(0, PChar(S), '', MB_OK);

  // Null 終端文字列を代入されると 0 までを文字列として認識します
  var A := PChar('Foo'#0); // Pascal は数字の前に # を付けると Char 型になる
  Writeln(A, ' ', Length(A)); // Foo 3 と表示される
end;
```

動的な型の導入

- **動的な型の導入**
  - 逆転の発想
  - 型に厳格なんだったら型さえあればいいんでしょ！
    - 「動的な型」っていう型を定義しちゃえばいい
    - 「型がない型」っていう型を定義しちゃうと？




```
begin
  // 動的な型 Variant
  var V: Variant := 'Foo';

  // 文字列が入っていた所に整数を入れられる
  V := 1;
  Writeln(V); // 1 が出力される
end.
```

```
// 引数の型を明示しなければ型無し型になる
procedure Foo(const A);
begin
  Writeln(Integer(A));
end;

begin
  var I := 1; // 型推論で Integer になる
  Foo(I);
end.
```

- **Variant (動的型)**
  - 大体なんでも入る型
    - レコード，集合，静的配列，クラス，クラス参照，およびポインタを除く
    - メモリを多く消費します
    - 処理速度は遅い
  - 型に厳格とはなんだったのか…

```
begin
  var V: Variant := 100;
  WriteLn(V);

  V := 'Foo';
  WriteLn(V);

  V := False;
  WriteLn(V);

  V := 12.5;
  WriteLn(V);

  V := False;
  var Foo: String := V; // 左辺の型に自動的に変換されます
  WriteLn(Foo);       // 文字列として False と表示されます
end.
```

- **TValue (動的型レコード)**
  - Variant の軽量版
  - 言語ではなく RTL で実装されています
    - System.Rtti
    - Variant は言語として実装されています
  - 型情報が手に入るので型情報を用いて処理を分岐したりできます

```
uses
 System.Rtti;
begin
  var A: TValue := 1; // 何でも代入できる
  Writeln(A.AsInteger);

  A := True;
  Writeln(A.ToString);

  if A.IsOrdinal then
    Writeln('Ordinal');
end.
```

- **const (型無し型)**
  - 型がない！
  - 使う時は型を指定します
    - 変数のアドレスが渡っています
    - 渡ってくる型を知っている必要があります

```
procedure Sample(const A);
begin
  try
    WriteLn(Integer(A));
  except on E: Exception do
    WriteLn(E.ToString);
  end;
end;

begin
  var P := 100;
  Sample(P); // 変数のアドレスを渡すため単純型の即値は渡せない
  Sample('あいうえお'); // 最初の 32 ビットが出る
end.
```

- **array of const (型無し型の配列)**
  - 型がな…ある！
  - array of const は array of TVarRec の糖衣構文
  - 様々な型の値を受取るために使われます
    - C の printf のような関数を作るために使われます
  - TVarRec は型情報を持っている record (共用体)
    - 定義は System.pas (RTL)

```
procedure Sample(A: array of const);
begin
  for var i := 0 to High(A) do
    if A[i].VType = vtInteger then // 型情報を使って整数型だけ表示する
    begin
      WriteLn('Integer: ', A[i].VInteger);
    end;
end;

begin
  Sample([5, 'あいうえお', True, nil]); // 5 だけ出力される
end.
```

まとめ

- **Object Pascal**
  - Pascal 由来の強い静的型付け言語
  - 型に対する柔軟さもある
    - 現実を見てる！
  - 他の言語にない珍しい機能もあります
    - メタクラスや動的配列、長い文字列型など
  - Delphi 10.4 Community Edition で Object Pascal の機能を試せます！
    - https://www.embarcadero.com/jp/products/delphi/starter
  - 最新版 Delphi 11.1 Alexandria では更に機能強化が入っています。
    - ２進数の即値や数値セパレータなど

End of document.

S/G シリアルゲームズ