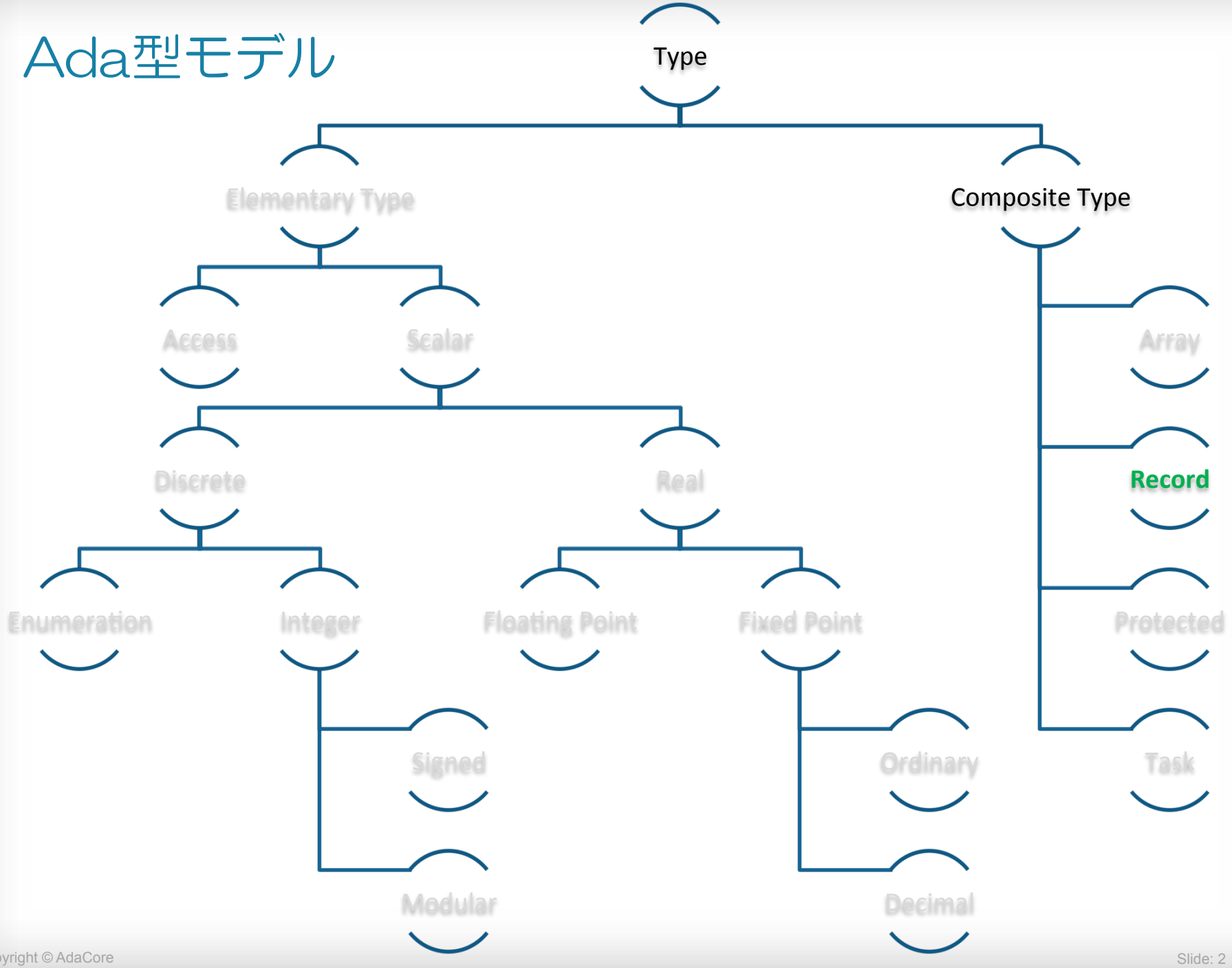
レコード型

Presented by Quentin Ochem

university.adacore.com

Ada型モデル



レコード型

- 一つの型の中に，名前を持つ異種のデータを格納することができます

```
type Shape is record
   Id : Integer;
   X, Y : Float;
end record;
```

- 各要素には，ドット表記を用いて，アクセスすることができます

```
   S : Shape;
begin
   S.X := 0.0;
   S.Id := 1;
```

ネスト化したレコード型

- 要素の型としては，全ての確定（definite）型を用いることができます

```
type Position is record
   X, Y : Integer;
end record;

type Shape is record
   Name : String (1 .. 10);
   P : Position;
end record;
```

- サイズは実行前準備処理時にのみ決定する場合があります

```
Len : Natural := Compute_Len;
type Name_Type is new String (1 .. Len);

type Shape is record
   Name : Name_Type;
   P : Position;
end record;
```

  - このとき生成するコードに影響を与えます

デフォルト値

- レコード型の要素に，デフォルト値を与えることができます

```
type Position is record
   X : Integer := 0;
   Y : Integer := 0;
end record;
```

- デフォルト値を動的に表現することもでき，オブジェクトの実行前準備処理で評価されます

```
   Cx, Cy : Integer := 0;

   type Position is record
      X : Integer := Cx;
      Y : Integer := Cy;
   end record;

   P1 : Position; -- = (0, 0);

begin

   Cx := 1;
   Cy := 1;

   declare

      P2 : Position; -- = (1, 1);
```

集成式（aggregate）(1/2)

- 配列と同様に，レコードの値を集成式の形式で与えることができます

```
type Position is record
   X, Y : Integer;
end record;

type Shape is record
   Name : String (1 .. 10);
   P : Position;
end record;

Center : Position := (0, 0);
Circle : Shape := ((others => ' '), Center);
```

- 名前付き集成式は利用可能です（しかし，途中で位置による指定に戻ることはできません）

```
P1 : Position := (0, Y => 0);       -- OK
P2 : Position := (X => 0, Y => 0); -- OK
P3 : Position := (Y => 0, X => 0); -- OK
P4 : Position := (X => 0, 0);       -- NOK
```

集成式(2/2)

- 要素が一つの場合，名前付き集成式を用いる必要があります

```
type Singleton is record
   V : Integer;
end record;

V1 : Singleton := (V => 0); -- OK
V2 : Singleton := (0);      -- NOK
```

- （不定としての）デフォルト値は，名前，あるいは「others」のあとで<>によって参照できます

```
type Rec is record
   A, B, C, D : Integer;
end record;

V1 : Rec := (others => <>);
V2 : Rec := (A => 0, B => <>, others => <>);
```

- もし，集成式中で，全ての残りの型が同じならば，「others」を用いることができます

```
type Rec is record
   A, B : Integer;
   C, D : Float;
end record;

V1 : Rec := (0, 0, others => 0.0);
```

区別子が必要な理由

- 使用する場面に応じて，以下に示すレコード型要素の一部のみを必要とするとします

```
type Shape is record
   X, Y : Float;
   X2, Y2 : Float;
   Radius : Float;
   Outer_Radius : Float;
end record;
```

- もし，形状（Shape）が，線分だとしたとき，半径（Radius）は必要でしょうか

区別子の利用

- 区別子を用いて，レコード型をパラメータ化することができます

```
type Shape_Kind is (Circle, Line, Torus);

type Shape (Kind : Shape_Kind) is record
   X, Y : Float;
   case Kind is
      when Line   =>
         X2, Y2 : Float;
      when Torus  =>
         Outer_Radius, Inner_Radius : Float;
      when Circle    =>
         Radius : Float;
   end case;
end record;
```

- この型は不定で，オブジェクトの宣言時に，「制約（constraint）」を加える必要があります

```
V : Shape (Circle);
```

区別子を持ったレコード型の利用

- 配列で，制約を持たない部分は記述しなければなりません

```
   V1 : Shape (Circle) := …;
   V2 : Shape := V1; -- OK, constrained by initialization
begin
   V1.Radius := 0.0; --  OK, radius is in the Circle case
   V2.X2 := 0.0;     --  Raises constraint error
```

- ある制約に関して，アクセスすることができない要素へのアクセスは，制約エラー（Constraint_Error）を送出します

- 注意: 区別子は定数です．オブジェクト宣言時に設定します

区別子を持った集成式

- レコードによる集成と同様です．しかし，区別子の値を与える必要があります

- 制約に関している値のみを与えます

```
V1 : Shape := (Kind => Line, X => 0.0, Y => 0.0, X2 => 10.0, Y2 => 10.0);
V2 : Shape := (Circle, 0.0, 0.0, 5.0);
```

Quiz

正しいですか (1/10)

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
type R is record
   A, B, C : Integer := 0;
end record;

V : R := (A => 1);
```

正しいですか (1/10)

いいえ

```
type R is record
   A, B, C : Integer := 0;
end record;

V : R := (A => 1);
```

コンパイルエラー．集成式では，他の要素に対しても
デフォルト値を与える必要があります
例えば，「A => 1, others => <>」です

正しいですか (2/10)

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
type My_Integer is new Integer;

type R is record
   A, B, C : Integer := 0;
   D : My_Integer := 0;
end record;

V : R := (others => 1);
```

正しいですか (2/10)

いいえ

```
type My_Integer is new Integer;

type R is record
   A, B, C : Integer := 0;
   D : My_Integer := 0;
end record;

V : R := (others => 1);
```

コンパイルエラー
全てのコンポーネントは同じ型ではありません
「others」によって，同じ値を与えることはできません

正しいですか (3/10)

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
type Cell is record
   Val : Integer;
   Next : Cell;
end record;
```

正しいですか (3/10)

いいえ

```
type Cell is record
   Val : Integer;
   Next : Cell;
end record;
```

コンパイルエラー
この型定義は，再帰しています

正しいですか (4/10)

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
type My_Integer is new Integer;

type R is record
   A, B, C : Integer;
   D : My_Integer;
end record;

V : R := (others => <>);
```

正しいですか (4/10)

はい

```
type My_Integer is new Integer;

type R is record
   A, B, C : Integer;
   D : My_Integer;
end record;

V : R := (others => <>);
```

これは正しい
レコード定義中で明示的に値が与えられていない場合，
A, B, C, D は，その時点でのメモリ中の値（何の値かは不定）になります

正しいですか (5/10)

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
type R is record
   A : Integer := 0;
end record;

V : R := (0);
```

正しいですか (5/10)

いいえ

```
type R is record
   A : Integer := 0;
end record;

V : R := (0);
```

コンパイルエラー
要素が一つの場合，名前付き表記のみ可能です（例えば，「A => 0」です）

正しいですか (6/10)

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
type R is record
   V : String;
end record;

V : R := (V => "Hello");
```

正しいですか (6/10)

いいえ

```
type R is record
   V : String;
end record;

V : R := (V => "Hello");
```

コンパイルエラー
レコード型は，制約を持たない要素を持つことができません

正しいですか (7/10)

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
type R is record
   S : String (1 .. 10);
end record;

V : R := (S => "Hello");
```

正しいですか (7/10)

いいえ

```
type R is record
   S : String (1 .. 10);
end record;

V : R := (S => "Hello");
```

実行時エラー（またコンパイル時に警告がでます）
ここでの文字列長は5です．しかし，長さ10の文字列が必要です

正しいですか (8/10)

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
   type R (D : Integer) is record
      null;
   end record;

   V1 : R := (D => 5);
   V2 : R := (D => 6);
begin
   V1 := V2;
```

正しいですか (8/10)

いいえ

```
   type R (D : Integer) is record
      null;
   end record;

   V1 : R := (D => 5);
   V2 : R := (D => 6);
begin
   V1 := V2;
```

V１とV2は異なる区別子の値を持ちます．
この両者は構造的に異なっていると考えます

正しいですか (9/10)

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
type Shape_Kind is (Circle, Line);

type Shape (Kind : Shape_Kind) is record
   case Kind is
      when Line   =>
         X, Y : Float;
         X2, Y2 : Float;
      when Circle =>
         X, Y : Float;
         Radius : Float;
   end case;
end record;
```

正しいですか (9/10)

いいえ

```
type Shape_Kind is (Circle, Line);

type Shape (Kind : Shape_Kind) is record
   case Kind is
      when Line   =>
         X, Y : Float;
         X2, Y2 : Float;
      when Circle =>
         X, Y : Float;
         Radius : Float;
   end case;
end record;
```

X，Yが，Line と Circle で重複しています

正しいですか　(10/10)

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
   type Shape_Kind is (Circle, Line);

   type Shape (Kind : Shape_Kind) is record
      X, Y : Float;
      case Kind is
         when Line   =>
            X2, Y2 : Float;
         when Circle =>
            Radius : Float;
      end case;
   end record;

   V : Shape := (Circle, others => <>);
begin
   V.Kind := Line;
   V.X2 := 0.0;
   V.Y2 := 0.0;
```

正しいですか (10/10)

いいえ

```
   type Shape_Kind is (Circle, Line);

   type Shape (Kind : Shape_Kind) is record
      X, Y : Float;
      case Kind is
         when Line   =>
            X2, Y2 : Float;
         when Circle =>
            Radius : Float;
      end case;
   end record;

   V : Shape := (Circle, others => <>);
begin
   V.Kind := Line;
   V.X2 := 0.0;
   V.Y2 := 0.0;
```

オブジェクトの区別子は定数です

university.adacore.com