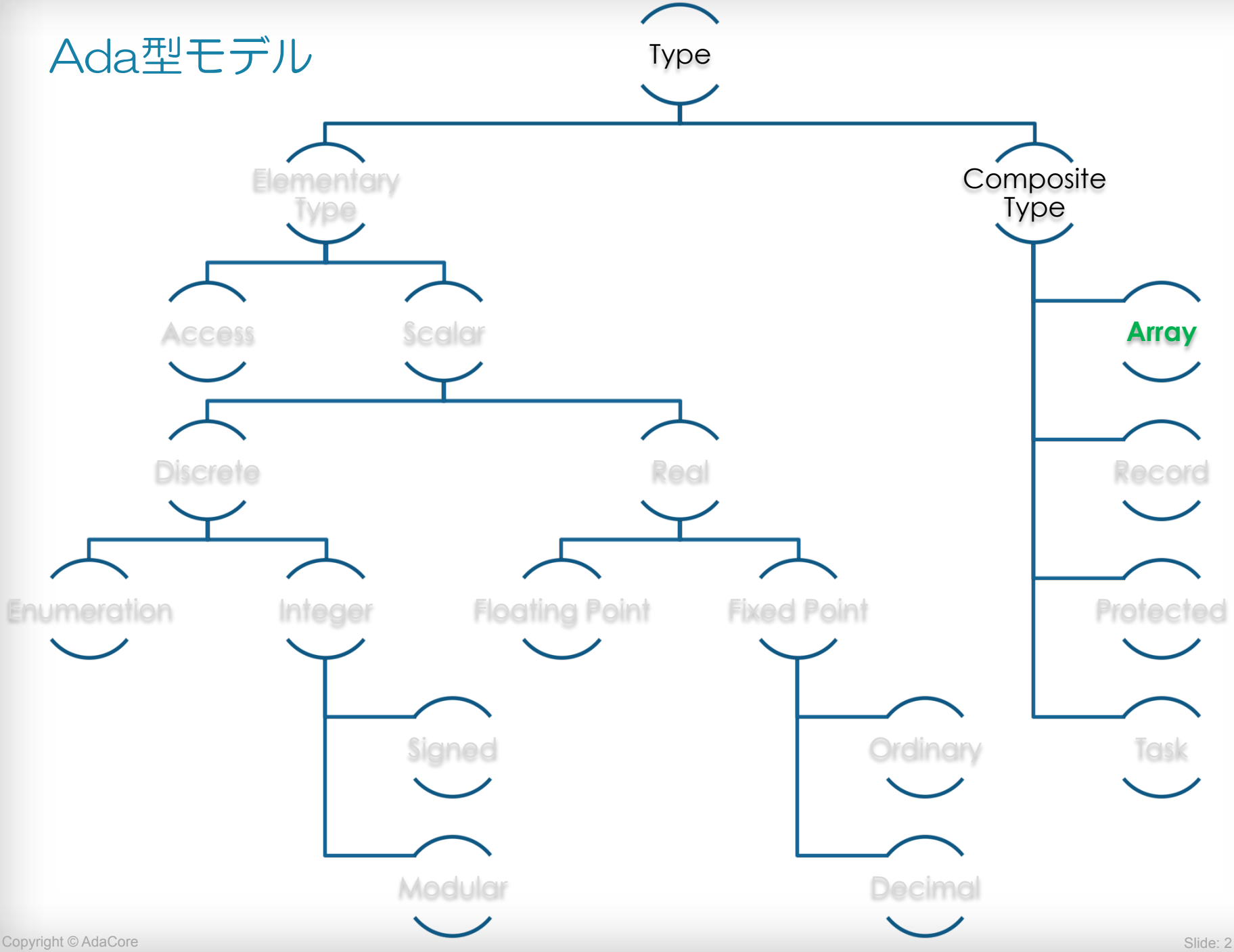

# 配列

**Presented Quentin Ochem**

university.adacore.com

# Ada型モデル



# 配列は，第一級オブジェクトである

- 全ての配列は（二重に）型を持つ

Ada

```
type T is array (Integer range <>)
 of Integer;

A : T (0 .. 14);
```

C++

```
int * A = new int [15];
```

- 配列型のプロパティは
  - 添字型（全ての離散型，明確な境界を持つ場合もある）
  - 要素型（全ての確定型）

- 配列オブジェクトのプロパティは
  - 配列型
  - 明確な境界
  - 明確な値

# 確定型と未定型

- 確定型（definite type）は，付加情報なしにオブジェクトを作ることができます
  - サイズが分かっている
  - 制約が分かっている
- 未定型（indefinite type）は，制約を加える必要があります
- 配列型は，両方可能です

```
type Definite is array (Integer range 1 .. 10) of Integer;
type Indefinite is array (Integer range <>) of Integer;

A1 : Definite;
A2 : Indefinite (1 .. 20);
```

- 配列型の要素は，確定している必要があります

# 配列の添字

- 全ての離散型は，配列の添字となることができます
  - 整数（符号付き或いはモジュラ型）
  - 列挙
- 配列添字は，連続する範囲上で定義することができます
- 配列添字の範囲が，「空」の場合があります

```
type A1 is array (Integer range <>) of Integer;
type A2 is array (Character range 'a' .. 'z') of Integer;
type A3 is array (Integer range 1 .. 0) of Integer;
type A4 is array (Boolean) of Integer;
```

- 配列添字は，配列宣言の時点で計算されます

```
X : Integer := 0;
type A is array (Integer range 1 .. X) of Integer;
--  changes to X don't change A instances after this point
```

# 配列要素にアクセスする

- 配列要素に直接アクセスすることができます

```
   type A is array (Integer range <>) of Integer;
   V : A (1 .. 10);
begin
   V (1) := 0;
```

- 配列型および配列オブジェクトでは次の属性を使用することができます；‘Length ‘Range ‘First ‘Last
- 要素へのアクセス時には，配列の持つ境界が動的に検査されます．範囲外へアクセスした場合，Constraint_Error 例外が送出されます

```
   type A is array (Integer range <>) of Float;
   V : A (1 .. 10);
begin
   V (0) := 0.0; -- NOK
```

# 配列のコピー

- 配列への操作は，「第一級操作」です

```
   type T is array (Integer range <>) of Integer;

   A1 : T (1 .. 10);
   A2 : T (1 .. 10);
begin
   A1 := A2;
```

- コピー操作において，配列長はチェックされますが，実添字はチェックされません

```
   type T is array (Integer range <>) of Integer;

   A1 : T (1 .. 10);
   A2 : T (11 .. 20);
   A3 : T (1 .. 20);
begin
   A1 := A2; -- OK
   A1 := A3; -- NOK
```

# 配列の初期化

- 配列を初期化するときに，コピーを行うことができます

```
type T is array (Integer range <>) of Integer;

A1 : T (1 .. 10);
A2 : T (11 .. 20) := A1;
```

- 未定型配列の場合，初期化時に演繹することで，範囲が定まります

```
type T is array (Integer range <>) of Integer;

A1 : T (1 .. 10);
A2 : T := A1; -- A2 bounds are 1 .. 10
```

# 部分配列（Array Slice）

- 配列の一部のみを（スライスとして）使用することができます
  - 一次元の配列に対してのみです

- 配列オブジェクトが必要となる場面で，この配列の部分（スライス）を使用することができます

```
   type T is array (Integer range <>) of Integer;

   A1 : T (1 .. 10);
   A2 : T (1 .. 20) := …;
begin
   A1 := A2 (1 .. 10);
   A1 (2 .. 4) := A2 (5 .. 7);
```

# 配列リテラル（集成式）

- 集成式で，配列全体に値を与えることができます

```
({[<position> => ] <expression>,} [others => <expression>])

(1, 2, 3)                    -- finite positional aggregate
(1 => 1, 2 => 10, 3 => 30)   -- finite named aggregate
(1, others => 0)             -- indefinite positional aggregate
(1 => 1, others => 0)        -- indefinite named aggregate
```

- 配列の値を必要とするどの場所でも利用できます
- 有限集成式は，変数における値の制約を初期化できます．範囲の下限は，T'Firstと等しくなります

```
   type T is array (Integer range <>) of Integer;

   V1 : T := (1, 2, 3);
   V2 : T := (others => 0); -- NOK (initialization)
begin
   V1 := (others => 0); -- OK (assignment)
```

# 配列の結合

- “&”演算子を用いることで，二つの一次元配列を結合することができます
  - 結合した配列の下限は，左被演算子の下限となります

```
type T is array (Integer range <>) of Integer;

A1 : T := (1, 2, 3);
A2 : T := (4, 5, 6);
A3 : T := A1 & A2;
```

- 配列は，値と結合することもできます

```
type T is array (Integer range <>) of Integer;

A1 : T := (1, 2, 3);
A2 : T := A1 & 4 & 5;
```

# 配列の同等性

- 2つの一元配列は，次の場合に等しくなります
  - 配列サイズが等しい
  - 対応する要素が等しい

```
   type T is array (Integer range <>) of Integer;

   A1 : T (1 .. 10);
   A2 : T (1 .. 20);

begin

   if A1 = A2 then -- ALWAYS FALSE
```

- 添字の実際の値は，配列の同等性に関しては，関係ありません

# 配列に対するループ

- 添字を用いたループ

```
type T is array (Integer range <>) of Integer;

A : T (1 .. 10);

for I in A'Range loop
   A (I) := 0;
end loop;
```

- オブジェクトを用いたループ

```
type T is array (Integer range <>) of Integer;

A : T (1 .. 10);

for V of A loop
   V := 0;
end loop;
```

# 行列

- 二次元配列

```
   type T is array (Integer range <>, Integer range <>) of Integer;
   V : T (1 .. 10, 0 .. 2);
begin
   V (1, 0) := 0;
```

- 属性：'First (次元), 'Last (次元), 'Range (次元)

- 配列の配列

```
   type T1 is array (Integer range <>) of Integer;
   type T2 is array (Integer range <>) of T1 (0 .. 2);
   V : T2 (1 .. 10);
begin
   V (1)(0) := 0;
```

# 文字列

- 文字列は，配列です

```
type String is array (Positive range <>) of Character;
```

- 特別な文字列リテラルがあります

```
V  : String := "This is it";
V2 : String := "Here come quotes ("")";
```

- ASCIIパッケージでは，文字の値に対して名前がついています

```
V  : String := "This is nul terminated" & ASCII.NUL;
```

# 配列要素のデフォルト値

- 通常，配列はデフォルト値を持ちません

```
   type T is array (Integer range <>) of Integer;
   V : T (1 .. 20);
begin
   Put_Line (Integer'Image (V (1))); -- Displays an uninitialized value
```

- 静的式を用いて，初期化時にデフォルト値を与えることは可能です（即ち，コンパイル時に値が定まります）

```
   type T is array (Integer range <>) of Integer
      with Default_Component_Value => 0;
   V : T (1 .. 20);
begin
   Put_Line (Integer'Image (V (1))); -- Displays 0
```

- 性能の面からは，上記の方法は望ましくありません

# 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 My_Int is new Integer range 1 .. 10;

   type T is array (My_Int) of Integer;

   V : T;
begin
   V (1) := 2;
```

# 正しいですか (1/10)

はい

```
   type My_Int is new Integer range 1 .. 10;

   type T is array (My_Int) of Integer;

   V : T;
begin
   V (1) := 2;
```

全て問題ありません

# 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 T is array (Integer) of Integer;

   V : T;
begin
   V (1) := 2;
```

# 正しいですか (2/10)

いいえ

```
   type T is array (Integer) of Integer;

   V : T;
begin
   V (1) := 2;
```

（構文的に）コンパイルには問題ありません．しかし，スタック領域に積まれるデータの量が大きいため失敗する可能性があります

# 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 T1 is array (Integer range <>) of Integer;
   type T2 is array (Integer range <>) of Integer;

   V1 : T1 (1 .. 3) := (others => 0);
   V2 : T2 := (1, 2, 3);
begin
   V1 := V2;
```

# 正しいですか (3/10)

いいえ

```
   type T1 is array (Integer range <>) of Integer;
   type T2 is array (Integer range <>) of Integer;

   V1 : T1 (1 .. 3) := (others => 0);
   V2 : T2 := (1, 2, 3);
begin
   V1 := V2;
```

コンパイルエラー．V1とV2は型が異なります

# 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 T is array (Integer range <>) of Integer;

   V : T := (1, 2, 3);
begin
   V (0) := V (1) + V (2);
```

# 正しいですか (4/10)

いいえ

```
   type T is array (Integer range <>) of Integer;

   V : T := (1, 2, 3);
begin
   V (0) := V (1) + V (2);
```

これら3つの添字の付け方が間違っています．Vの添字範囲は，
Integer'First .. Integer'First + 2 となります

# 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 T is array (Integer range <>) of Integer;

   V1 : T (1 .. 2);
   V2 : T (10 .. 11) := (others => 0);
begin
   V1 := V2;
```

# 正しいですか (5/10)

はい

```
   type T is array (Integer range <>) of Integer;

   V1 : T (1 .. 2);
   V2 : T (10 .. 11) := (others => 0);
begin
   V1 := V2;
```

全て問題ありません．（V1の）各要素の値は，V2の値をスライドさせることで正しい添字に与えられます

# 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）

```
   X : Integer := 10;
   type T is array (Integer range 1 .. X) of Integer;
   V1 : T;
begin
   X := 100;
   declare
      V2 : T;
   begin
      V1 := V2;
```

# 正しいですか (6/10)

はい

```
   X : Integer := 10;
   type T is array (Integer range 1 .. X) of Integer;
   V1 : T;
begin
   X := 100;
   declare
      V2 : T;
   begin
      V1 := V2;
```

問題ありません．Xの値が変化するときでさえ，配列の宣言は影響を受けません．
V1とV2は，同じ境界を持ちます

# 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 T is array (Integer range <>) of Integer;
   V1 : T (1 .. 3) := (10, 20, 30);
   V2 : T := (10, 20, 30):
begin
   for I in V1'Range loop
      V1 (I) := V1 (I) + V2 (I);
   end loop;
```

# 正しいですか (7/10)

いいえ

```
   type T is array (Integer range <>) of Integer;
   V1 : T (1 .. 3) := (10, 20, 30);
   V2 : T := (10, 20, 30):
begin
   for I in V1'Range loop
      V1 (I) := V1 (I) + V2 (I);
   end loop;
```

V2(I) は例外を送出します．範囲がV1の範囲と異なっているからです

# 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 T is array (Integer range 1 .. 10) of Integer;

V : T (2 .. 9);
```

# 正しいですか (8/10)

いいえ

```
type T is array (Integer range 1 .. 10) of Integer;

V : T (2 .. 9);
```

Tの境界は固定されています．オブジェクト宣言中で変更することはできません

# 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）

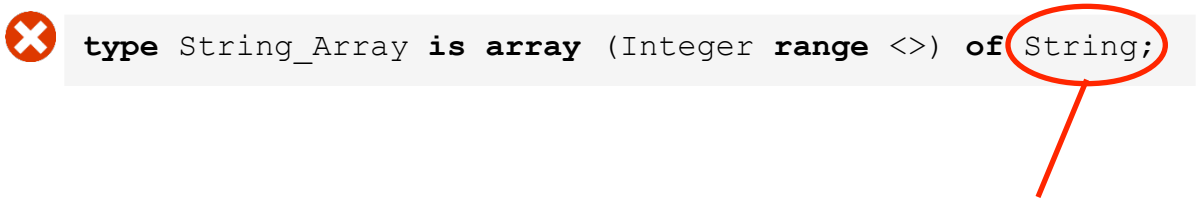
いいえ
（エラーの場所をクリックする）

```
type String_Array is array (Integer range <>) of String;
```

# 正しいですか (9/10)

いいえ

```
type String_Array is array (Integer range <>) of String;
```



配列は制限されたオブジェクトのみを含むことができます

# 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）

```
X : Integer := 0;

type T is array (Integer range <>) of Integer
   with Default_Component_Value => X;

V : T (1 .. 10);
```

# 正しいですか (10/10)

いいえ

```
X : Integer := 0;

type T is array (Integer range <>) of Integer
   with Default_Component_Value => X;

V : T (1 .. 10);
```

Xは静的式ではありません.
（コンパイル時に値が不明です）


university.adacore.com