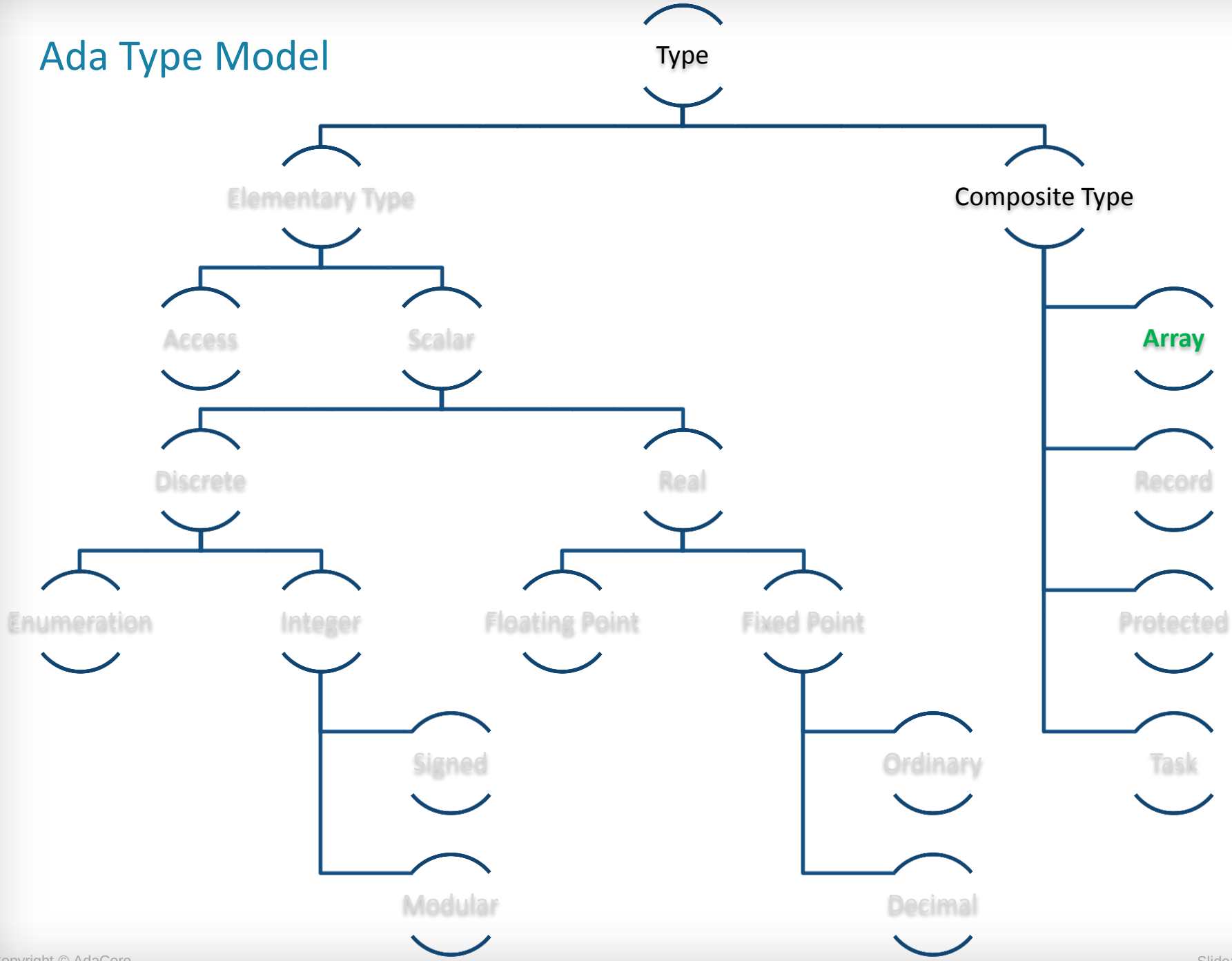
Arrays

Presented Quentin Ochem

university.adacore.com

Ada Type Model



Arrays are First Class Citizens

- **All arrays are (doubly) typed**

Ada

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

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

C++

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

- **Arrays types properties are**
  - The index type (can be any discrete type, with optional specific boundaries)
  - The component type (can be any definite type)

- **Arrays objects properties are**
  - The array type
  - Specific boundaries
  - Specific values

Definite vs. Indefinite Types

- **Definite types are types that can be used to create objects without additional information**
  - Their size is known
  - Their constraints are known
- **Indefinite types need additional constraint**
- **Array types can be definite or indefinite**

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

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

- **Components of array types must be definite**

Array Indices

- **Array indexes can be of any discrete type**
  - Integer (signed or modular)
  - Enumeration
- **Array indexes can be defined on any contiguous range**
- **Array index range may be empty**

```
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;
```

- **Array indexes are computed at the point of array declaration**

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

Accessing Array Components

- **Array components can be directly accessed**

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

- **Array types and array objects offer ‘Length, ‘Range, ‘First and ‘Last attributes**
- **On access, bounds are dynamically checked and raise Constraint_Error if out of range**

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

Array Copy

- **Array operations are first class citizens**

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

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

- **In copy operations, lengths are checked, but not actual indices**

```
   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
```

Array Initialization

- **Array copy can occur at initialization time**

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

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

- **If the array type is of an indefinite type, then an object of this type can deduce bounds from initialization**

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

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

Array Slices

- **It's possible to consider only a part of the array using a slice**
  - For array with only one dimension

- **Slices can be used where an array object is required**

```
   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);
```

Array Literals (Aggregates)

- **Aggregates can be used to provide values to an array as a whole**

```
({[<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
```

- **They can be used wherever an array value is expected**
- **Finite aggregate can initialize variable constraints, lower bound will be equal to T'First**

```
   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)
```

Array Concatenation

- **Two 1-dimensional arrays can be concatenated through the "&" operator**
  - The resulting array lower bound is the lower bound of the left

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

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

- **An array can be concatenated with a value**

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

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

Array Equality

- **Two 1-dimensional arrays are equal if**
  - Their size is equal
  - Their components are equal one by one

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

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

begin

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

- **Actual indexes do not matter in array equality**

Loops over an Array

- **Through an index loop**

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

A : T (1 .. 10);

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

- **Through an object loop**

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

A : T (1 .. 10);

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

Matrices

- **Two dimensional arrays**

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

- **Attributes are 'First (dimension), 'Last (dimension), 'Range (dimension)**

- **Arrays of arrays**

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

Strings

- **Strings are regular arrays**

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

- **There is a special String literal**

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

- **The package ASCII provides names for Character values**

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

Default Component Values

- **By default, arrays do not come with default values**

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

- **It's possible to force a default value initialization through a static expression (value known at compile-time)**

```
   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
```

- **For performance reasons, the above may not be desirable**

Quiz

Is this correct? (1/10)

YES
(click on the check icon)

NO
(click on the error location(s))

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

   type T is array (My_Int) of Integer;

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

Is this correct? (1/10)

YES

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

   type T is array (My_Int) of Integer;

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

Everything is OK

Is this correct? (2/10)

YES
(click on the check icon)

NO
(click on the error location(s))

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

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

Is this correct? (2/10)

NO

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

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

**This can actually compile fine,
but will fail at compilation due to the
amount of data reserved on the stack**

Is this correct? (3/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
   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;
```

Is this correct? (3/10)

NO

```
   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;
```

Compilation error, V1 and V2 are not of the same type

Is this correct? (4/10)

YES
(click on the check icon)

NO
(click on the error location(s))

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

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

Is this correct? (4/10)

NO

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

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

These three accesses are wrong, V is constrained between Integer'First .. Integer'First + 2

Is this correct? (5/10)

YES
(click on the check icon)

NO
(click on the error location(s))

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

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

Is this correct? (5/10)

YES

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

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

Everything is OK. The assignement will slide the values of V2 in the correct indexes.

Is this correct? (6/10)

YES
(click on the check icon)

NO
(click on the error location(s))

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

Is this correct? (6/10)

YES

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

No problem. Even if the value of X changes, the array declaration is not impacted
V1 and V2 have the same boundaries

Is this correct? (7/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
   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;
```

Is this correct? (7/10)

NO

```
   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) will raise an exception because the range is different from the one of V1

Is this correct? (8/10)

YES
(click on the check icon)

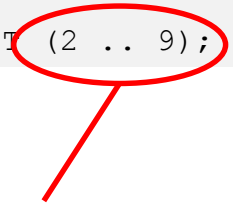
NO
(click on the error location(s))

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

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

Is this correct? (8/10)

NO

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

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



The boundaries of T are fixed, can't be changed at object declaration

Is this correct? (9/10)

YES
(click on the check icon)

NO
(click on the error location(s))

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

Is this correct? (9/10)

NO

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

Array can only contain constrained objects

Is this correct? (10/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
X : Integer := 0;

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

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

Is this correct? (10/10)

NO

```
X : Integer := 0;

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

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

X is not a static expression (unknown at compile-time)

university.adacore.com