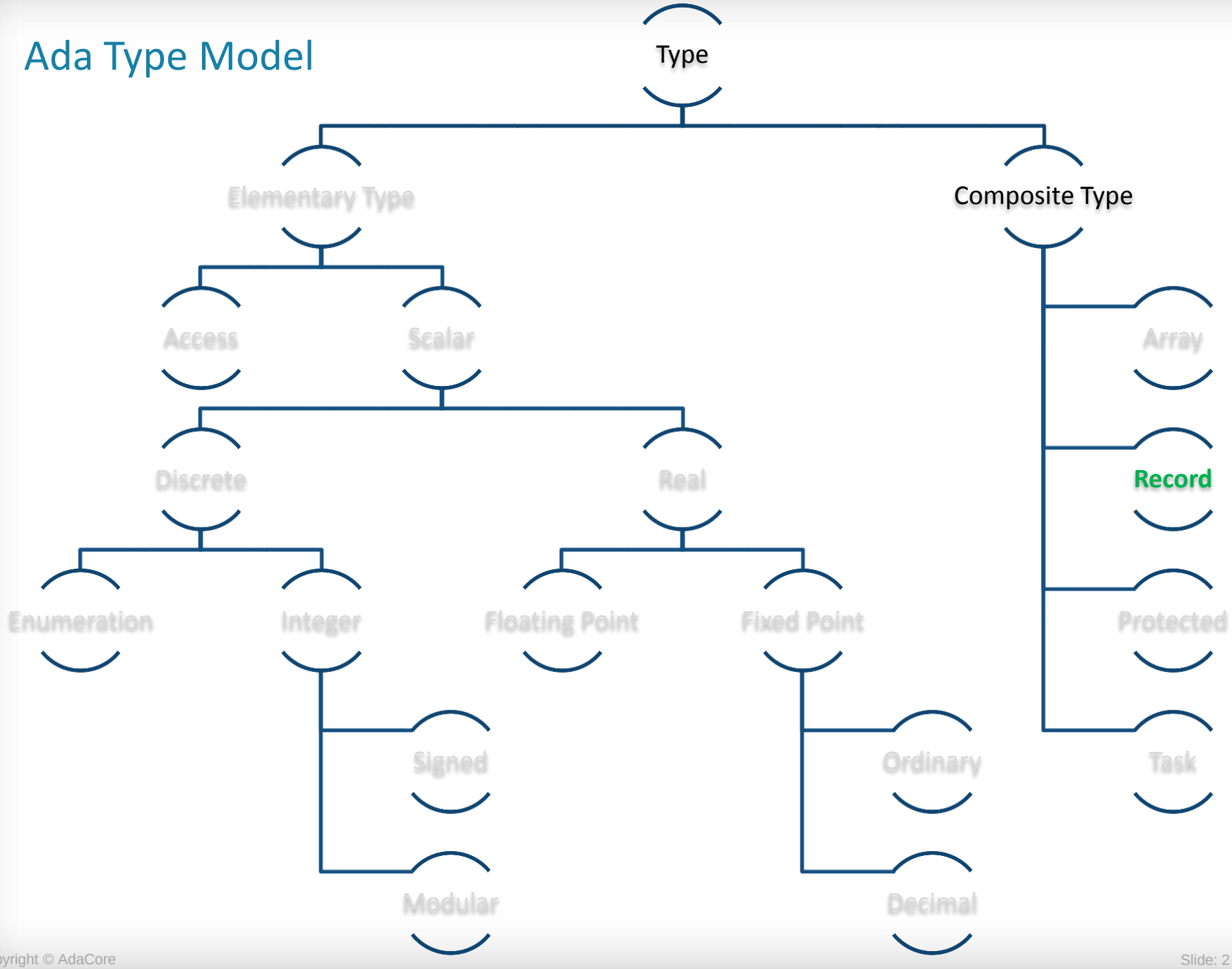
Record Types

Presented by Quentin Ochem

university.adacore.com

Ada Type Model



Record Types

- **Allow to store named heterogeneous data in a type**

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

- **Fields are accessed through dot notation**

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

Nested Record Types

- **Any kind of definite type can be used as component types**

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

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

- **Size may only be known at elaboration time**

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

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

  - Has impact on code generated

Default Values

- **Default Values can be provided to record components:**

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

- **Default values are dynamic expressions evaluated at object elaboration**

```
   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);
```

Aggregates (1/2)

- **Like arrays, record values can be given through aggregates**

```
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);
```

- **Named aggregates are possible (but cannot switch back to positional)**

```
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
```

Aggregates (2/2)

- **Named aggregate is required for one-elements records**

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

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

- **Default values can be referred as <> after a name or others**

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

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

- **If all remaining types are the same, others can use an expression**

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

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

Discriminant Rationale

- **Only a subset of the components are needed to use this type, depending on the context**

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

- **Why do we need to use the memory for Radius if the shape is a line?**

Use of a Discriminant

- **Types can be parameterized after a discrete type**

```
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;
```

- **This type is indefinite, need to be constrained at object declaration**

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

Usage of a Record with Discriminant

- **As for arrays – the unconstrained part has to be specified**

```
   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
```

- **Accessing to a component not accessible for a given constraint will raise *Constraint_Error***

- **Note: A discriminant is constant, set at object declaration**

Aggregates with Discriminants

- **Same as record aggregates – but have to give a value to the discriminant**
- **Only the values related to the constraint are supplied**

```
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

Is this correct? (1/10)

YES
(click on the check icon)

NO
(click on the error location(s))

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

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

Is this correct? (1/10)

NO

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

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

Compilation error, the aggregate should give a default value for other fields, for example (A => 1, others => <>)

Is this correct? (2/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
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);
```

Is this correct? (2/10)

NO

```
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);
```

Compilation error, all components are not of the same type, they can't be given a common value through others

Is this correct? (3/10)

YES
(click on the check icon)

NO
(click on the error location(s))

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

Is this correct? (3/10)

NO

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

Compilation error, this type definition is recursive

Is this correct? (4/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
type My_Integer is new Integer;

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

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

Is this correct? (4/10)

YES

```
type My_Integer is new Integer;

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

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

This is correct. In the absence of explicit values given in the record definition, A, B, C and D will be of whatever value is in the memory at this time

Is this correct? (5/10)

YES
(click on the check icon)

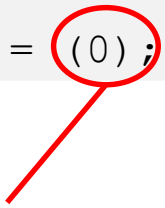
NO
(click on the error location(s))

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

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

Is this correct? (5/10)

NO

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

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



Compilation error, only the named notation is allowed in singleton values, e.g. (A => 0)

Is this correct? (6/10)

YES
(click on the check icon)

NO
(click on the error location(s))

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

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

Is this correct? (6/10)

NO

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

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

Compilation error, a record can't have an unconstrained component

Is this correct? (7/10)

YES
(click on the check icon)

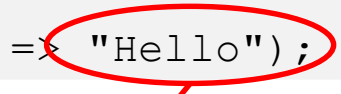
NO
(click on the error location(s))

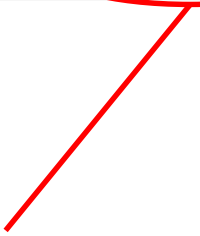
```
type R is record
   S : String (1 .. 10);
end record;

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

Is this correct? (7/10)

NO

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

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





Run-time error (and possible a compile-time warning).
The size of the string given here is 5, while we expect a string of 10 elements

Is this correct? (8/10)

YES
(click on the check icon)

NO
(click on the error location(s))

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

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

Is this correct? (8/10)

NO

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

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

V1 and V2 have different discriminant values, they're considered structurally different

Is this correct? (9/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
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;
```

Is this correct? (9/10)

NO

```
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 and Y components are duplicated

Is this correct? (10/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
   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;
```

Is this correct? (10/10)

NO

```
   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;
```

The discriminant of an object is constant

university.adacore.com