AdaML

“ Ada-tailored UML Modeling Language ”

Francesc Rocher

August 4, 2019

What is AdaML?

- A modeling language to draw UML diagrams
- UML tailored for Ada 2012 programming language
- Small, extensible tool implemented on top of PlantUML

What is *not* AdaML?

- A model-based tool to generate Ada code
- A reverse engineering tool to draw UML diagrams from existing Ada code
- An GUI-based UML modeling or drawing tool

Features

- Coherent set of functions to describe components
- Syntax resembling Ada
- High quality drawings (`eps`) to embed in other docs
- `svg` graphics for additional editing (e.g., inkscape)
- Online generated diagrams for online docs (e.g., GitHub)
- Easy to learn by example, both `AdaML` and Ada language

Online Version - Quick Start

- Open PlantUML Previewer or PlantText editor
- Remove default lines and paste the following code:

```
!include https://raw.github.com/rocher/AdaML/release/0.1.0/AdaML.puml
begin_type("Pan_Dimensional")
  procedure("Ask_The_Question", "in out Natural")
end()

begin_package("Deep_Thought")
  function("Answer_The_Question", "", "Natural")
private()
  variable("The_Answer", "Natural", 42)
end()

depends("Pan_Dimensional", "Deep_Thought", "ask >")
```

Online Version - Test Examples

In case you want to test the examples in this doc

- Visit one of the several available PlantUML online servers
- Copy and paste the `AdaML` code shown in the examples
- Replace the line '`!include AdaML.puml`' with

```
!include https://raw.github.com/rocher/AdaML/release/0.1.0/AdaML.puml
```

Warning

Depending on the PlantUML version used in the web site you can obtain different graphical results

Local Version - Requirements

In case you want to integrate results in your local docs

- `PlantUML` installed and working in your system
- `AdaML.puml` file, check `AdaML` releases
- Your favorite text editor with `PlantUML` support (e.g. Emacs)
- Check the list of supported editors
- For better visualization and integration with LaTeX and other formats, computer modern fonts

Pros & Cons

Online Version

- ✓ Quick and easy to test
- ✓ Same code can be embedded in online docs
- ✻ Bitmap graphics; `svg` in PlantText
- ✗ No file saved, no backups
- ✗ PlantUML version may change without notice, affecting graphical results, layout, style or other incompatibilities

Local Version

- ✓ Best graphical results to embed in other docs (`eps`)
- ✓ Consistent graphical results over time, if no version changed
- ✗ Requires PlantUML up and running

Document Conventions

In this document

Entities are *classifiers*, in UML terminology, that correspond to Ada packages, types, records, arrays, subtypes, tasks and protected objects

Elements are *features*, in UML terminology, that correspond to Ada record members, functions and procedures

Specifiers are Ada constructions, like type discriminant or range specifications (e.g, `range 1..1024`) that can appear in some entity diagrams

OOP Support

- `AdaML` supports both OOP and non-OOP approaches to software models
- There is no explicit difference between them in `AdaML`
- The final implementation in Ada is the last choice between the two options
- This document and examples on it use OOP concepts as much as possible

Abstraction Levels

`AdaML` provides two main view levels, according to the C4 model:

Code view to show in detail how entity elements (attributes, methods) are implemented and the relationship they have with other entities

Component view to show internal entity building blocks and the overall relationship with other entities

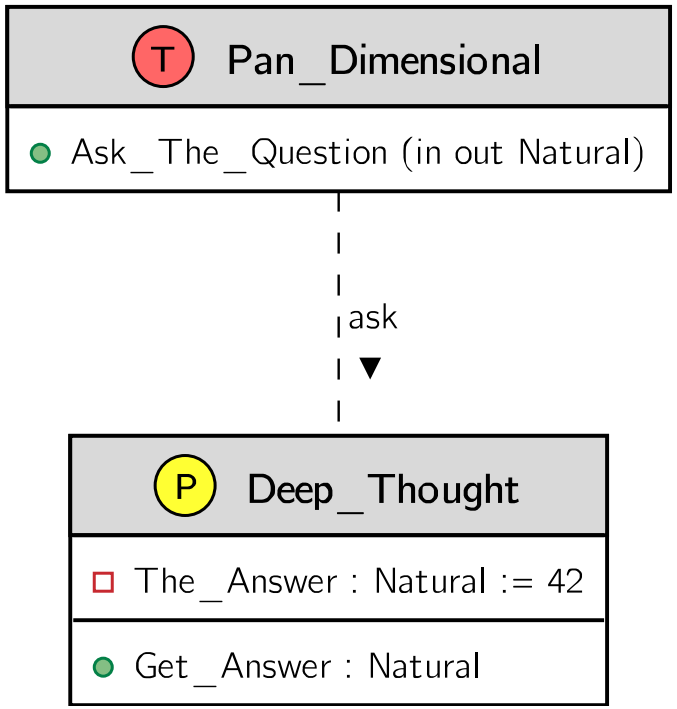
In some *code views*, Ada code is included.

Deep Thought Example - Code View

AdaML

```
!include AdaML.puml
begin_type("Pan_Dimensional")
  procedure("Ask_The_Question", "in out Natural")
end()

begin_package("Deep_Thought")
  function("Get_Answer", "", "Natural")
private()
  variable("The_Answer", "Natural", 42)
end()

depends("Pan_Dimensional", "Deep_Thought", "ask >")
```

UML



View Levels

Minimal only shows the package entity and generic parameters

Code View shows package elements with certain level of detail; elements can be other packages, types subprograms, tasks, constants, variables and protected objects

Component View shows package elements and their relationships; do not include details of such elements

Minimal view

The simplest package representation is

AdaML

```
!include AdaML.puml
package("Deep_Thought")
```

UML

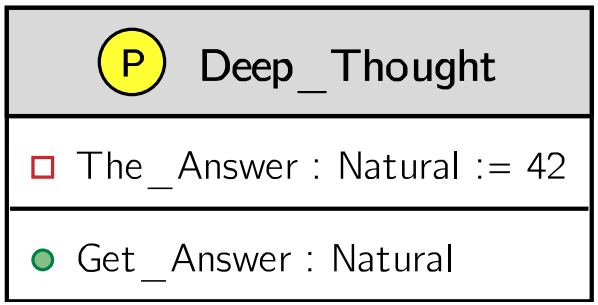
Code View

Shows package elements; here variables and subprograms shown

AdaML

```
!include AdaML.puml
begin_package("Deep_Thought")
  function("Get_Answer", "", "Natural")
private()
  variable("The_Answer", "Natural", 42)
end()
```

UML



Ada

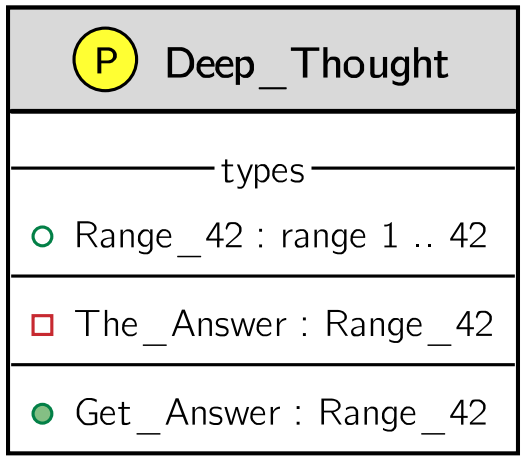
```
package Deep_Thought is
  function Get_Answer return Natural;
private
  The_Answer : Natural := 42;
end FooBar;
```

Adding Types

AdaML

```
!include AdaML.puml
begin_package("Deep_Thought")
  type("Range_42", "range 1 .. 42")
private()
  variable("The_Answer", "Range_42")
public()
  function("Get_Answer", "Range_42")
end()
```

UML



Note

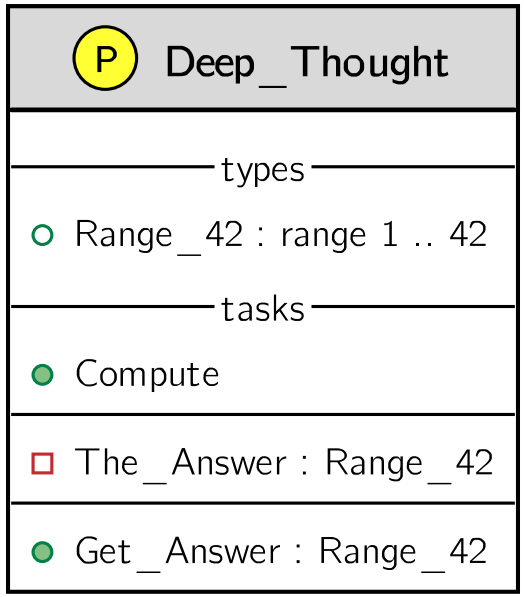
When using extra *entity elements*, like `types` or `tasks`, then variables and methods must be manually sorted (contrast with previous slide)

Adding Tasks

AdaML

```
!include AdaML.puml
begin_package("Deep_Thought")
  type("Range_42", "range 1 .. 42")
  task("Compute")
  variable("-The_Answer", "Range_42")
  function("Get_Answer", "", "Range_42")
end()
```

UML



Element visibility

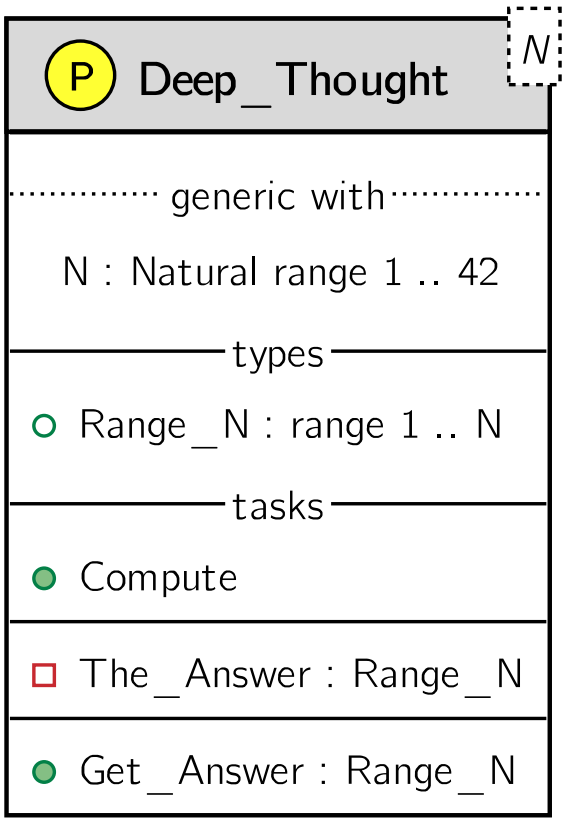
- use `private()` and `public()` functions, as in previous slides
- put '-', '+' or '#' in front of the element's name; this has priority over pri/pub functions

Generic Packages

AdaML

```
!include AdaML.puml
begin_package("Deep_Thought<N>")
  generic_with("N : Natural range 1 .. 42")
  type("Range_N", "range 1 .. N")
  task("Compute")
  variable("-The_Answer", "Range_N")
  function("Get_Answer", "", "Range_N")
end()
```

UML

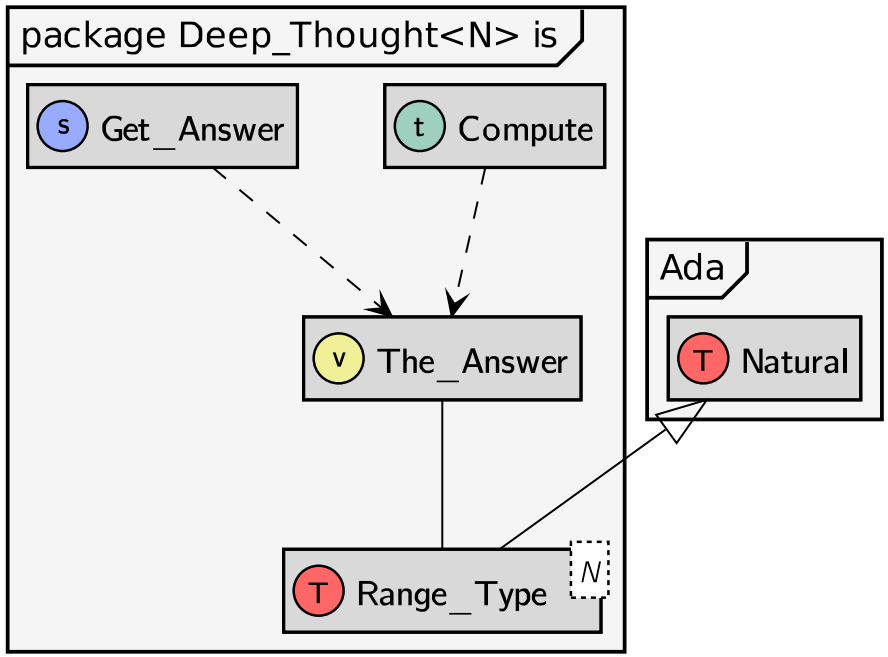


Component view

AdaML

```
!include AdaML.puml
begin_package_spec("Deep_Thought<N>")
  type("Range_Type<N>")
  type_new("Ada.Natural", "Range_Type")
  task("Compute")
  subprogram("Get_Answer")
  variable("The_Answer")
  depends("Get_Answer", "The_Answer");
  depends("Compute", "The_Answer");
  is("The_Answer", "Range_Type")
end()
```

UML



Type Classification

Ada 2012 overall type classification

```
ELEMENTARY TYPES
  |-- Scalar
  |   |-- Discrete
  |   |   |-- Integer
  |   |   |   |-- Signed
  |   |   |   '-- Modular
  |   |   '-- Enumeration
  |   '-- Real
  |       |-- Float
  |       '-- Fixed
  |           |-- Decimal
  |           '-- Ordinary
  '-- Access
```

```
COMPOSITE TYPES
  |-- Record
  |-- Array
  |-- Protected
  '-- Task
```

Predefined Types

- `AdaML` is aware of `Boolean`, `Integer`, `Natural`, `Positive`, `Float`, `Character` and `String` types
- so there is no need to declare such types

- When needed, refer to them also as, e.g. `Ada.Boolean`[1], to make them appear in a separate package
- otherwise it might look like the type has been redefined in your package

[1] This is UML notation context, not Ada programming language

Discrete Types

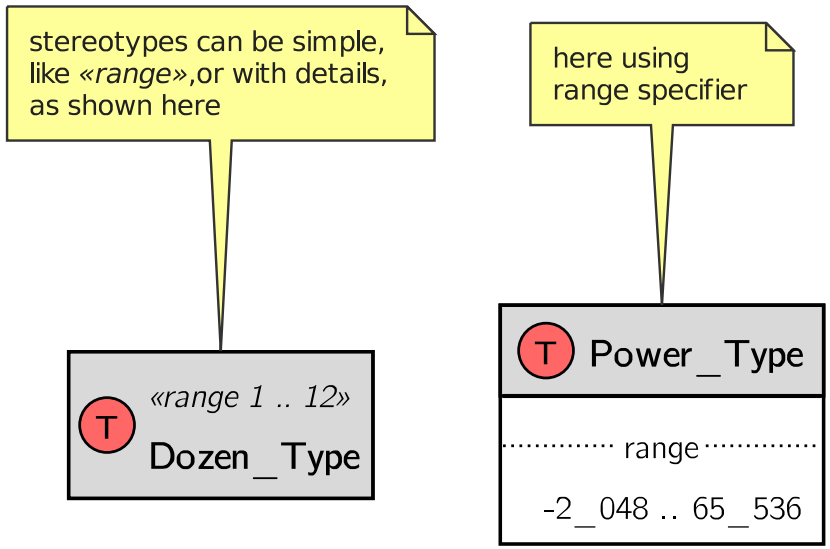
- For discrete types like *range*, *modular* or *enumeration* types, use either the stereotype or the particular specifier
- The meaning is the same, the appearance changes substantially
- Prefer stereotypes in component views, and specifiers in code views
- Both the stereotype and the specifier can be also combined; might be little redundant, but sometimes useful

Range Types

AdaML

```
!include AdaML.puml

' with stereotype
type("Dozen_Type", "range 1 .. 12")
note("top of Dozen_Type", "ste...")

' with specifier
begin_type("Power_Type")
  range("-2_048 .. 65_536")
end()
note("top of Power_Type", "her...")
```

UML



Ada

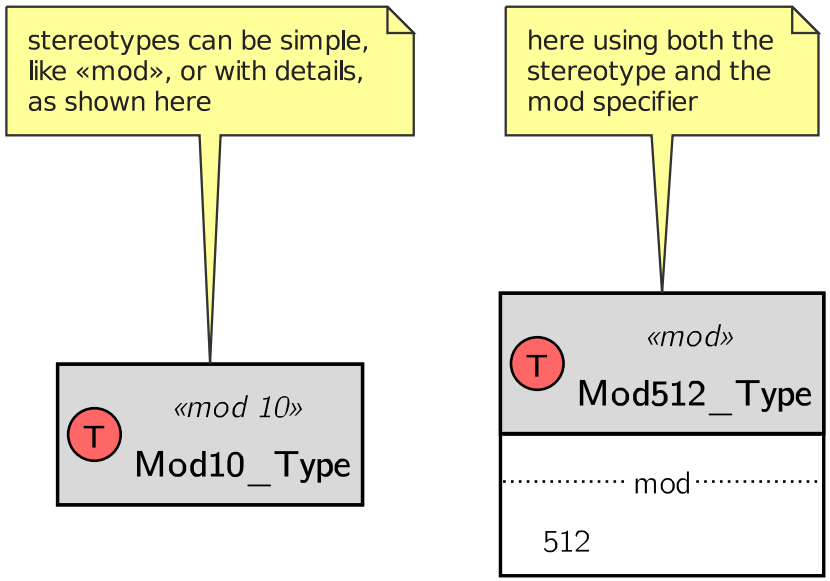
```
type Dozen_Type is range 1 .. 12;
type Power_Type is range -2_048 .. 65_536;
```

Modular Types

AdaML

```
!include AdaML.puml

' with stereotype
type("Mod10_Type", "mod 10")
note("top of Mod10_Type", "ste...")

' with mod specifier
begin_type("Mod512_Type", "mod")
  mod(512)
end()
note("top of Mod512_Type", "he...")
```

UML



Ada

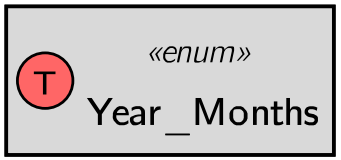
```
type Mod10_Type is mod 10;
type Mod512_Type is mod 512;
```

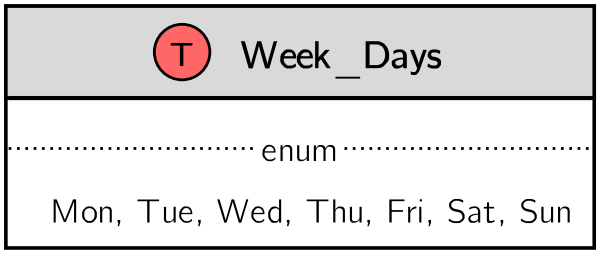
Enumeration Types

AdaML

```
!include AdaML.puml

' with stereotype
type("Year_Months", "enum")

' with specifier
begin_type("Week_Days")
  enum("Mon, Tue, Wed, Thu, Fri, Sat, Sun")
end()

top_down("Year_Months", "Week_Days")
```

UML





Ada

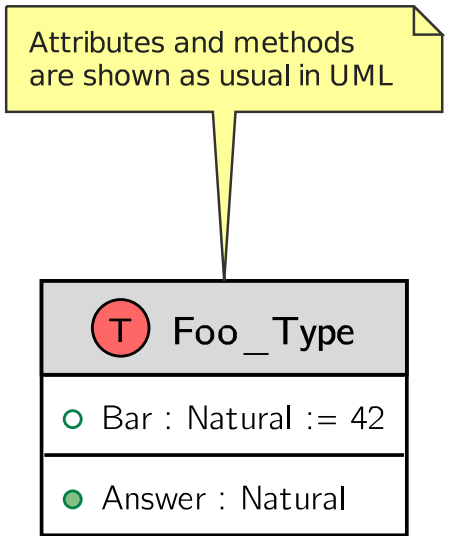
```
type Year_Months is (Jan, Feb, Mar, ..., Dec );
type Week_Days is (Mon, Tue, Wed, Thu, Fri, Sat, Sun);
```

Record Types

AdaML

```
!include AdaML.puml
begin_type("Foo_Type")
  function("Answer", "", "Natural")
  attribute("Bar", "Natural")
end()
```

UML



Ada

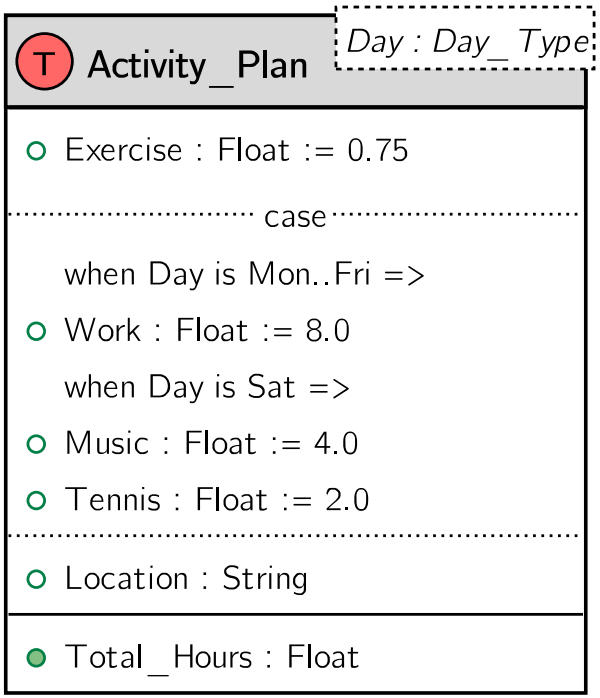
```
type Foo_Type is private record
  Baz : Natural;
end record;
function Answer (Arg : in out Foo_Type) return Natural;
```

Record Types with Discriminant (i)

AdaML

```
!include AdaML.puml
begin_type("Activity_Plan<Day : Day_Type>")
  attribute("Exercise", "Float := 0.75")
  case("Day is Mon..Fri")
    attribute("Work", "Float := 8.0")
  case("Day is Sat")
    attribute("Music", "Float := 4.0")
    attribute("Tennis", "Float := 2.0")
  case()
  attribute("Location", "String")
  function("Total_Hours", "", "Float")
end()
```

UML



Record Types with Discriminant (ii)

- When a case value is not specified, assume `null`

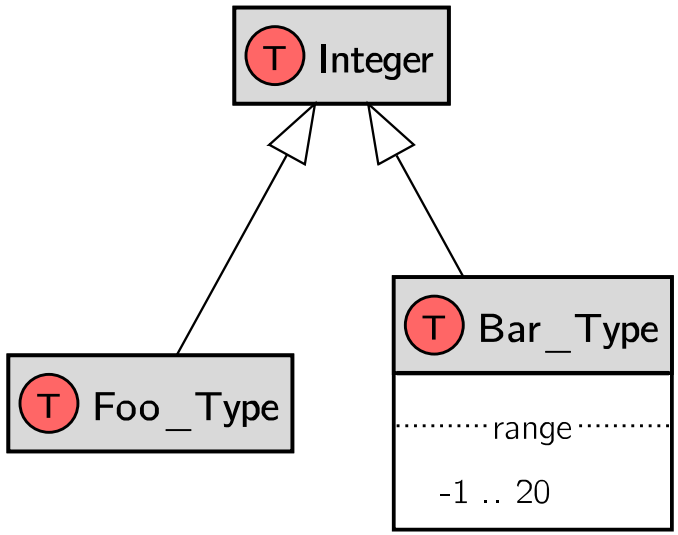
```
type Day_Type is (Mon, Tue, Wed, Thu, Fri, Sat, Sun);
type Activity_Plan (Day : Day_Type) is record
   Exercise : Float := 0.75;
   case Day is
      when Mon .. Fri =>
         Work : Float := 8.0;
      when Sat =>
         Music : Float := 4.0;
         Tennis : Float := 2.0;
      when Sun =>
         null;
   end case;
   Location : String;
end record;

function Total_Hours (Plan : in Activity_Plan) return Float;
```

Derived Types

AdaML

```
!include AdaML.puml
type_new("Integer", "Foo_Type")
begin_type_new("Integer", "Bar_Type")
  range("-1 .. 20")
end()
```

UML



Ada

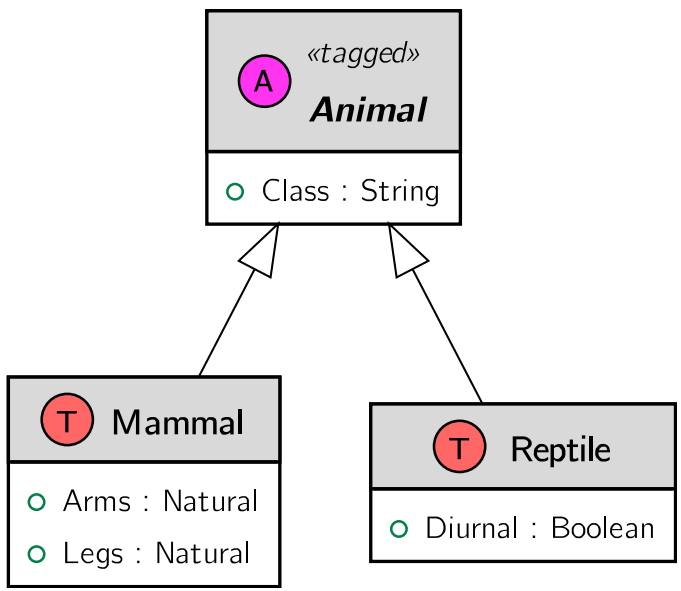
```
type Foo_Type is new Integer;
type Bar_Type is new Integer range -1 .. 20;
-- or simply
type Bar_Type is range -1 .. 20;
```

Abstract Types

AdaML

```
!include AdaML.puml
begin_abstract("Animal", "tagged")
  attribute("Class", "String")
end()

begin_type_new("Animal", "Mammal")
  attribute("Arms", "Natural")
  attribute("Legs", "Natural")
end()

begin_type_new("Animal", "Reptile")
  attribute("Diurnal", "Boolean")
end()
```

UML

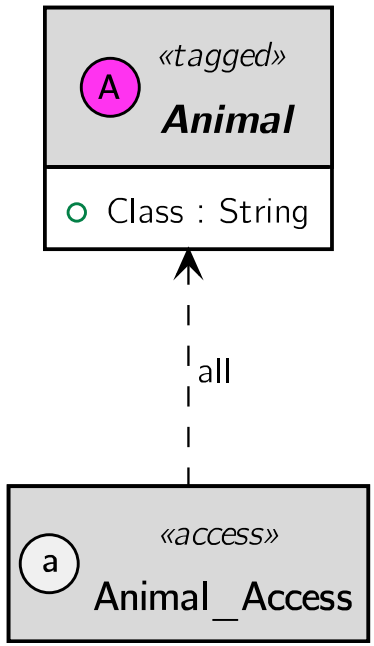


Access Types

AdaML

```
!include AdaML.puml
begin_abstract("Animal", "tagged")
  attribute("Class", "String")
end()

type_access("Animal")
```

UML



Note

Suffix '_Access' is automatically added to access types; suffix can be configured or suppressed

Introduction

- Notes are usually needed to clarify some UML constructions
- AdaML notes can be
  - *floating*, not associated to any entity
  - *linked* to one entity, appeared some slides before
  - *named notes* linked simultaneously to several entities
  - *notes on links*, associated to links between entities
- AdaML supports geometry hints to specify relative position between the note and the entity

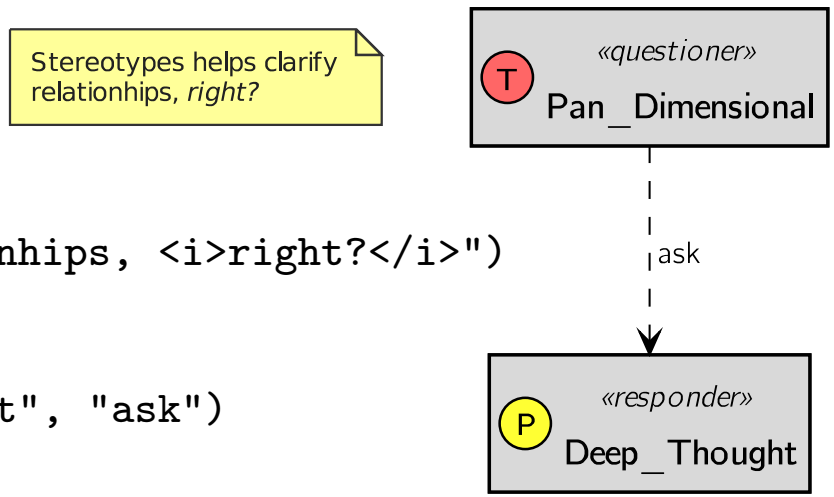
Floating Notes

- Not associated to any particular entity

AdaML

```
!include AdaML.puml
note("Stereotypes help clarify\nrelationhips, <i>right?</i>")
type("Pan_Dimensional", "questioner")
package("Deep_Thought", "responder")
depends("Pan_Dimensional", "Deep_Thought", "ask")
```

UML



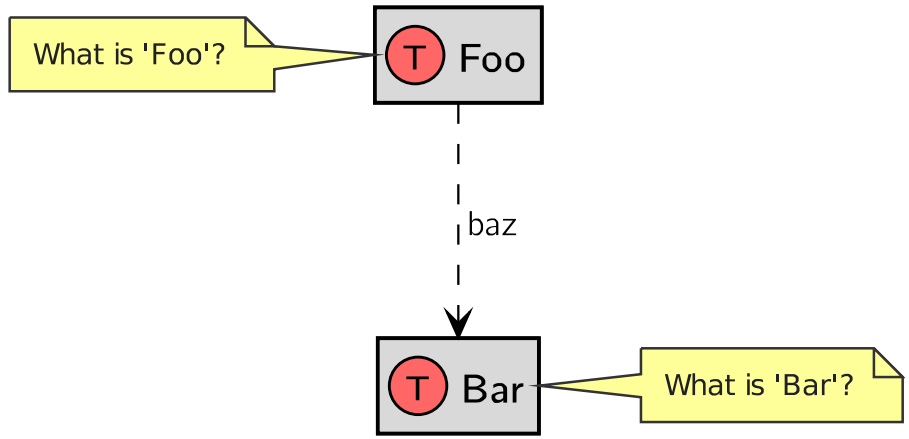
Notes Linked to Latest Defined Entity

- Notes are automatically associated to the latest defined entity
- Must specify relative position with ‘top’, ‘bottom’, ‘left’ or ‘right’ keyword

AdaML

```
!include AdaML.puml
type("Foo")
note("left", "What is 'Foo'?")

type("Bar")
note("right", "What is 'Bar'?")

depends("Foo", "Bar", "baz")
```

UML



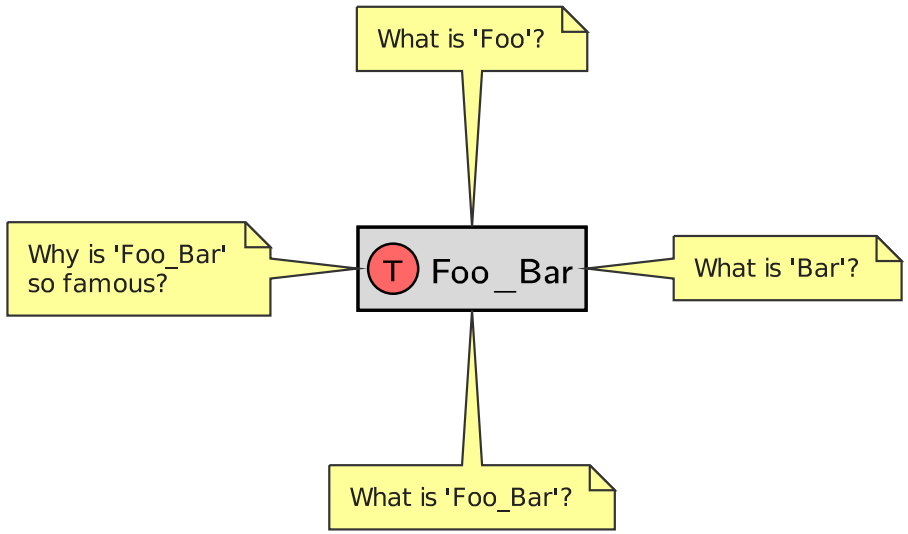
Notes Linked to an Explicit Entity

- Must specify relative position with ‘`top of`’, ‘`bottom of`’, ‘`left of`’ or ‘`right of`’ keyword, followed by the name of the entity

AdaML

```
!include AdaML.puml

type("Foo_Bar")
note("top of Foo_Bar", "W...?")
note("right of Foo_Bar", "W...?")
note("bottom of Foo_Bar", "W...?")
note("left of Foo_Bar", "Why...?")
```

UML



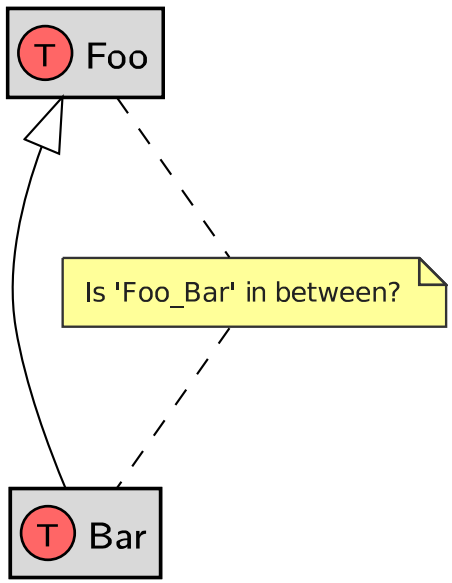
Named Notes Linked to Several Entities

- Specify a number, e.g. 42, to create the named note ‘Note_42’, and then link the note to other entities

AdaML

```
!include AdaML.puml
type("Foo")
type_new("Foo", "Bar")

note(42, "Is 'Foo_Bar' in between?")
link("Foo", "Note_42")
link("Note_42", "Bar")
```

UML



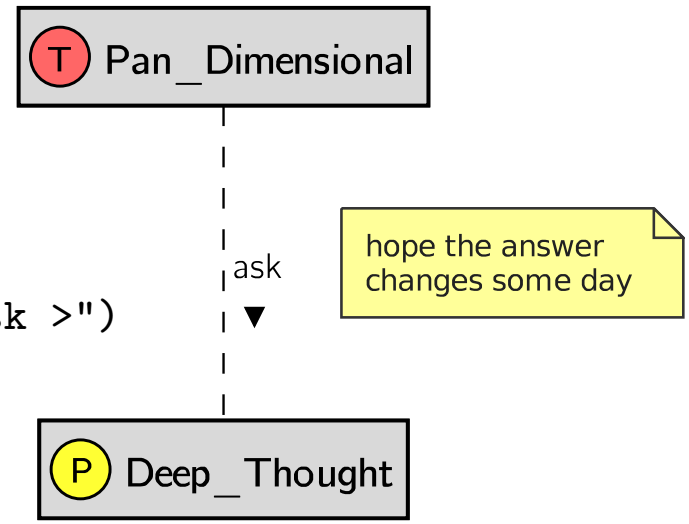
Notes on Links

- Automatically associated to the latest relationship defined
- Must contain ‘`on link`’ keyword, optionally preceded by ‘`top`’, ‘`bottom`’, ‘`left`’ or ‘`right`’ keyword

AdaML

```
!include AdaML.puml
type("Pan_Dimensional")
package("Deep_Thought")

depends("Pan_Dimensional", "Deep_Thought", "ask >")
note("right on link", "hope..")
```

UML



Release

Status

- AdaML is available under the terms of the GPLv3 License
- Current release is `r0.1.0`

ChangeLog

- First public release including everything you see in this overview document

TODO

Static behavior

- There are still a number of features to document and under development, like relationships, protected objects, task entries, arrays, etc..

Dynamic behavior

- Next releases will include interaction diagrams like sequence diagrams, activity diagrams, state machine diagrams or timing diagrams

Feedback Welcome

- Send comments, feedback or suggestions using the issues page of the AdaML GitHub repository: http://github.com/rocher/AdaML/issues
- Remember to specify the release number you are referring to in you comments
- Don't hesitate to contact the author in case you need help