Fortran classes and objects

Victor Eijkhout, Susan Lindsey

COE 322 Fall 2021

1. Classes and objects

Fortran classes are based on `type` objects, a little like the analogy between C++ `struct` and `class` constructs.

New syntax for specifying methods.

2. Object is type with methods

You define a type as before, with its data members, but now the type has a `contains` for the methods:

```
Module multmod

  type Scalar
     real(4) :: value
   contains
     procedure,public :: &
          printme,scaled
  end type Scalar

contains ! methods
  !! ...
end Module multmod
```

```
Program Multiply
  use multmod
  implicit none

  type(Scalar) :: x
  real(4) :: y
  x = Scalar(-3.14)
  call x%printme()
  y = x%scaled(2.)
  print '(f7.3)',y

end Program Multiply
```

3. Method definition

```
subroutine printme(me)
  implicit none
  class(Scalar) :: me
  print '("The value is",f7.3)',me%value
end subroutine printme
function scaled(me,factor)
  implicit none
  class(Scalar) :: me
  real(4) :: scaled,factor
  scaled = me%value * factor
end function scaled
```

4. Class organization

- You're pretty much forced to use `Module`
- A class is a `Type` with a `contains` clause
  followed by `procedure` declaration
- Actual methods go in the `contains` part of the module
- $\Rightarrow$ First argument of method is the object itself. $\Leftarrow$

5. Similarities and differences

	C++	Fortran
Members	in the object	in the 'type'
Methods	in the object	interface: in the type implementation: in the module
Constructor	default or explicit	none
object itself	'this'	first argument
Class members	global variable	accessed through first arg
Object's methods	period	percent

6. Point program

```
Module PointClass
  Type,public :: Point
     real(8) :: x,y
   contains
     procedure, public :: &
        distance
  End type Point
contains
  !! ... distance function ...
  !! ...
End Module PointClass
```

```
Program PointTest
  use PointClass
  implicit none
  type(Point) :: p1,p2

  p1 = point(1.d0,1.d0)
  p2 = point(4.d0,5.d0)

  print *,"Distance:",&
      p1%distance(p2)

End Program PointTest
```

Exercise 1

Take the point example program and add a distance function:

```
Type(Point) :: p1,p2
! ... initialize p1,p2
dist = p1%distance(p2)
! ... print distance
```

Exercise 2

Write a method add for the `Point` type:

```
Type(Point) :: p1,p2,sum
! ... initialize p1,p2
sum = p1%add(p2)
```

What is the return type of the function add?