

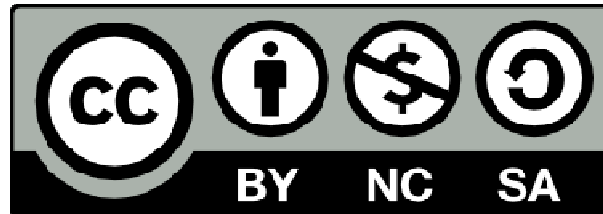


# Fortran classes and data visibility

---



# Reusing this material



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

[http://creativecommons.org/licenses/by-nc-sa/4.0/deed.en\\_US](http://creativecommons.org/licenses/by-nc-sa/4.0/deed.en_US)

This means you are free to copy and redistribute the material and adapt and build on the material under the following terms: You must give appropriate credit, provide a link to the license and indicate if changes were made. If you adapt or build on the material you must distribute your work under the same license as the original.

Note that this presentation may contain images owned by others. Please seek their permission before reusing these images.



# Classes

- Extends derived types
  - Introduces concept of type-bound procedures
  - Class methods

```
module building
  implicit none
  integer, parameter :: MAXLEN = 100
  type person
    character(MAXLEN) :: name
    integer :: officeNumber
  contains
    procedure :: getName
    procedure :: setName
    procedure :: getOfficeNumber
    procedure :: setOfficeNumber
  end type person
end module building
```

## Person

```
name: String
officeNumber: Integer
```

```
getName(): String
setName(String): Boolean
getOfficeNumber(): Integer
setOfficeNumber(Integer)
```



# Type bound procedure

```
PROCEDURE [(interface-name)] [[, binding-  
attr-list ]::] binding-name[=> procedure-  
name]
```

binding-attr-list:

- PASS, NOPASS
- NON\_OVERRIDABLE
- DEFERRED
- PUBLIC, PRIVATE



# Visibility

- Recall, derived type by default public
- Can make data and procedures default private using the `private` keyword
  - For procedures keyword comes after `contains`
- Explicitly can set procedures:
  - `private`
  - `public`



# Visibility example

```
module building
  implicit none
  private
  integer, parameter :: MAXLEN = 100
  type person
    private
    character(MAXLEN) :: name
    integer :: officeNumber
  contains
    private
    procedure, public :: getName
    procedure, public :: setName
    procedure, public :: getOfficeNumber
    procedure, public :: setOfficeNumber
  end type person
end module building
```



# Class variable

- Type bound procedures must take a class variable
  - Variable name is not prescribed (self is not a keyword)
  - Automatically passed
  - Allows for data polymorphism

```
...  
contains  
function getName(self)  
class(person), intent(inout):: self  
character(MAXLEN) :: getName  
    getName = self%name  
end subroutine
```

```
...  
end module building  
• Could then be used:  
type(person) :: bob  
...  
write(*,*) bob%getName()  
...
```



# Unlimited type

- Allowed unlimited polymorphic type

```
class (*)
```

- Pass in any type of variable or object
- Enables truly polymorphic routines
  - Combine with type-guarding for useful functionality

- If allocatable

- Either type needs specified:

```
class (*), allocatable :: fred  
allocate(person::fred)
```

- Or source type needs specified:

```
person :: bob  
class (*), allocatable :: fred  
allocate(fred, source=bob)
```

- In this case the allocation is made and the values copied into the new object





# Select type

- Type inquiry/type guarding is possible
- `type is`
  - Type of object is the specified type
- `class is`
  - Class of the object is the same as the specified class or an extension of that class

```
select type (bob)
```

```
type is (manager)
```

```
  print *, 'This is a manager'
```

```
class is (person)
```

```
  print *, 'This could be a manager or person'
```

```
class default
```

```
  print *, 'Unknown type used'
```

```
end select
```



# Type comparison functions

- Two new intrinsic functions to inquire about types:

`EXTENDS_TYPE_OF (X, Y)`

- Returns true if the type of X is the same as, or extends the type of Y
- Some subtleties if Y is unallocated unlimited polymorphic type

`SAME_TYPE_AS (X, Y)`

- Returns true if the type of Y is the same as the type of X



# Class constructor

- Can specify a constructor
  - Using interface with same name as the derived type

...

```
public :: person
type person
  character(MAXLEN) :: name
  integer :: officeNumber
contains
  procedure, public :: getName
  procedure, public :: setName
  procedure, public :: getOfficeNumber
  procedure, public :: setOfficeNumber
end type person
interface person
  module procedure initialise_person
end interface
```

- Can be overloaded
- Not mandatory



# Class destructor

- `final` keyword can be used to define procedure(s) to be called on object destruction

```
public :: person
  type person
    character(MAXLEN) :: name
    integer :: officeNumber
  contains
    procedure, public :: getName
    procedure, public :: setName
    procedure, public :: getOfficeNumber
    procedure, public :: setOfficeNumber
    final :: cleanUp
  end type person
  interface person
    module procedure initialise_person
  end interface
```



# Summary

- F2003 allows tying procedures to derived types
  - Creates true classes
- Class procedures, by default, pass the class as an argument
- Default visibility of data and procedures public
  - Can easily restrict to make object safer and more object like
- Constructors and destructors available



# Exercise

- Convert your basic derived types into classes by adding type bound procedures
- Explore unlimited polymorphism to build procedures that can work on different data types
- Do the same with percolate

