UK Atomic Energy Authority

UKAEA

Teaching an old dog new tricks: Object-oriented programming in Fortran

Chris MacMackin

RSECon 2019

Motivation

- Frequently need to work with legacy code
- Need modern techniques to be productive
- One option is using Object Oriented Programming features available in Fortran
- Aim to introduce these little known features
- Will consider where they are useful (and not)

Contents

- Intro to object oriented programming
- OOP features in Fortran
- Comparison to other languages
- Case studies
- Conclusions

Object oriented programming

- Paradigm that groups code and data
- Define new data types—*classes*
- Create instances of classes—*objects*
- Call routines attached to objects—*methods*
- Seen in languages such as C++, Java, Python...

Principles of OOP

- Encapsulation (information hiding)
  - Data can be made private to class, preventing interference and hiding implementation details
- Inheritance
  - A class can extend a previous one, replicating its behaviour, creating an “is-a” hierarchy
- Polymorphism
  - Code does not depend on exact type of object

Fortran data types

- FORTRAN77 had limited types: integer, real, logical, etc.
- Fortran 90 introduced *derived types* (binding multiple pieces of data together)

```fortran
module type_foo_mod
  type :: foo
    real, private :: real_comp
    integer, public :: int_comp
  end type foo
end module type_foo_mod
```

```fortran
program foo_example
  use type_foo_mod
  type(foo) :: f1

  f1%int_comp = 42
  f1%real_comp = 3.14 ! ILLEGAL
  ! Can only access private
  ! component in defining module
end program foo_example
```

OOP in Fortran

- 2003 standard extended derived types to provide OOP features
- Can now define *type-bound procedures*
- Types can inherit from others
- Polymorphic variables (indicated with “class”) can hold a type or its subtype

Code example

```
module oop_example_mod
  type :: parent
    integer, private :: i = 5
  contains
    procedure :: get_val=>get_parent
  end type parent

  type, extends(parent) :: child
  contains
    procedure :: get_val=>get_child
  end type child
contains
  integer function get_parent(this)
    class(parent), intent(in):: this
    get_parent = this%i
  end function

  integer function get_child(this)
    class(child), intent(in):: this
    get_child = 2*this%i - 1
  end function
end module oop_example_mod
```

```
program polymorphism_demo
  use oop_example_mod
  type(parent) :: p
  type(child) :: c

  call print_val(p) ! Prints 5
  call print_val(c) ! Prints 9

contains

  subroutine print_val(obj)
    class(parent), intent(in) :: obj
    print*, obj%get_val()
  end subroutine print_val

end program polymorphism_demo
```

Interfaces/abstract types

- Can define *deferred* methods which are not implemented
- Implementation left to inheriting types
- Allows definition of an interface

```
module interface_example_mode
  type, abstract :: differentiator
  contains
    procedure(diff), deferred :: &
          derivative
  end type differentiator

  abstract interface
    function diff(this, f, dx)
      class(differentiator), &
            intent(in) :: this
      real, intent(in) :: f(:)
      real, intent(in) :: dx
      real, dimension(size(f)) &
                       :: diff
    end function diff
  end interface

end module interface_example_mod
```

Other features

- Overloaded type-bound procedures
- Procedure pointer components
- Non-overridable type-bound procedures
- Finalizers (destructors)
- Overloaded arithmetic, logic, IO operators
- Parameterized derived types (limited generic programming)

Comparison to C++

- In both
  - Static typing
  - Methods implemented outside class definition
  - Can place implementations in separate file (with *submodules*)

- Some differences
  - Fortran only has single-inheritance
  - Methods are *virtual* by default
  - To overload methods, must specifically define them as *generic*

Comparison to Python

- Like Python:
  - Methods just routines that take object as first argument
  - Objects passed by reference

- Fortran differs b/c
  - Methods implemented outside of class definition
  - Components of derived type fixed at compile-time
  - Static typing, etc...

Compiler support

- Vendors were slow to implement OOP in Fortran
- Now supported by most vendors, but can be buggy
- E.g., GNU has memory leak issues
- Be prepared to file bug reports
- Optimisation needs work (e.g., Alam 2014)

Limitations

- Fortran OOP less powerful than in other languages
  - Can’t call methods on function results

```
name = linked_list%get_element(3)%get_name()  ! Illegal
```

  - **No generic programming**
  - Limited ability to overload
  - Single inheritance, no concept of interfaces
- Very verbose syntax
- Sometimes easier to wrap Fortran with Python

Case studies

- OOP can be very useful, but not a panacea
- In some cases it makes life simpler, in others makes it more complex
- Comes with a performance overhead
- Will look at two clearly beneficial uses, one where the benefits are less clear

Wrapping legacy code

- Can define a class for making calls to old F77-style procedures
- Class can manage creation of work arrays
- Store arguments as type components
- E.g., wrapping a solver from LAPACK
  - Define a type representing a matrix
  - It holds LU factorisation, work arrays
  - Call a “solve” method for vector of data

Strategy pattern

- Define an abstract class for some task, with different implementations in subclasses
- Makes it easy to add new features later
- Can offer a choice of algorithms
  - E.g., an integrator class with subtypes for Runge-Kutta, Adams-Bashforth, etc.
- Also useful for parameterising physics
  - E.g., different equations of state for a gas

Abstract calculus

- Create abstract type with methods needed by a solver (e.g. add, multiply, derivative)
- Can then write mathematical solvers that are agnostic about details of the problem
- This is elegant in principle, but
  - Needs lots of boilerplate code
  - Can’t use existing array-based packages
  - Compiler bugs can cause memory leaks

Takeaways

- Fortran now has modern features like OOP
- These are a natural extension of Fortan 90
- Can be useful for dealing with new and legacy code
- However, less powerful and clunkier syntax than in other languages
- Not suitable to all problems, but worth considering

Other resources

- Code from this presentation: https://github.com/cmacmackin/OOP-Fortran-Examples
- Introduction to OOP in the 2003 standard: https://wg5-fortran.org/N1601-N1650/N1648.pdf
- Information on compiler support: http://fortranwiki.org/fortran/show/Compiler+Support+for+Modern+Fortran
- Examples of using OOP: https://github.com/sourceryinstitute/Scientific-Software-Design
- Is Fortran still Relevant (Shahid Alam, 2014): https://arxiv.org/pdf/1407.2190

Thank you

Any questions?