Object Oriented Programming with Fortran

An Overview

Carlos Cruz

NASA GSFC Code 606 (ASTG)
Greenbelt, Maryland 20771

carlos.a.cruz@nasa.gov

October 25, 2018

Agenda

Object Oriented Programming with Fortran

- Object Oriented Programming
  - Background
- OOP Features in Fortran 2003
  - Data Abstraction
  - Encapsulation
  - Inheritance
  - Polymorphism
- Examples

Programming paradigms

- **Procedural**
  - C, Fortran90
    - Focus on writing good functions and procedures
    - Computation changes the program state
- **Functional**
  - Lisp, Haskell
    - Emphasizes use of state-less functions
- **Object Oriented**
  - Smalltalk, Java
    - Programs manipulate objects
    - Objects have an internal state
- **Multi-paradigm**
  - C++, Python, Fortran2003

Object Oriented Programming (OOP)

OOP is a major paradigm shift. It grew out of perceived weaknesses of *structured programming*:

- Modifications are difficult/expensive.
- Developers need to be expert in all parts of the application.
- Limited modularity.

encapsulation

Object Oriented Programming (OOP)

OOP is a major paradigm shift. It grew out of perceived weaknesses of *structured programming*:

- Modifications are difficult/expensive.
- Developers need to be expert in all parts of the application.
- Limited modularity.
- Centralized development constraint.

encapsulation
inheritance

Object Oriented Programming (OOP)

OOP is a major paradigm shift. It grew out of perceived weaknesses of *structured programming*:

- Modifications are difficult/expensive.
- Developers need to be expert in all parts of the application.
- Limited modularity.
- Centralized development constraint.

- Multiple implementations of the same functionality.

encapsulation
inheritance

polymorphism

Object Oriented Programming (OOP)

OOP is a major paradigm shift. It grew out of perceived weaknesses of *structured programming*:

- Modifications are difficult/expensive.
- Developers need to be expert in all parts of the application.
- Limited modularity.
- Centralized development constraint.

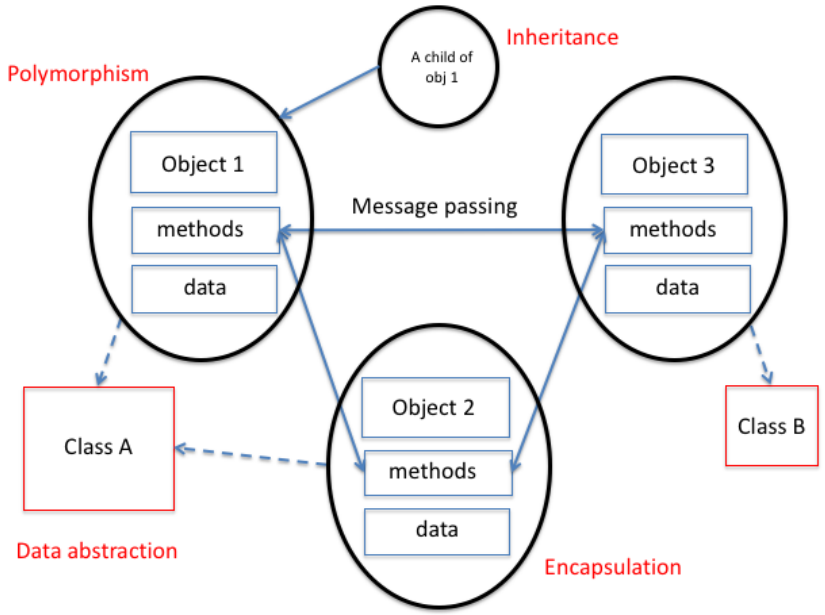
encapsulation
inheritance

- Multiple implementations of the same functionality.
- Need to support several data structures that are nearly identical but vary in some systematic ways.
- Difficult to maintain consistency as such structures are extended.

polymorphism
templates

What is OOP?

OOP is a paradigm in which a program's state and behavior are bundled into **objects**.



- A **class** is a data type
  - Attributes
  - Behaviors
- An **Object** is an **instance** of a class.
  - Behavior of objects is expressed in terms of methods which are the class procedures. Methods have privileged access to object state.
  - Method invocation may look different than regular procedure calls.

Within a program, objects interact with each other by sending messages (i.e. invoking methods)

Caveats

- OOP is a major paradigm shift which generally takes years to fully absorb.
- We hope to *motivate* the rationale for using OO Fortran in some circumstances.

How do we write Fortran programs using the OO paradigm? What OOP support does Fortran provide?

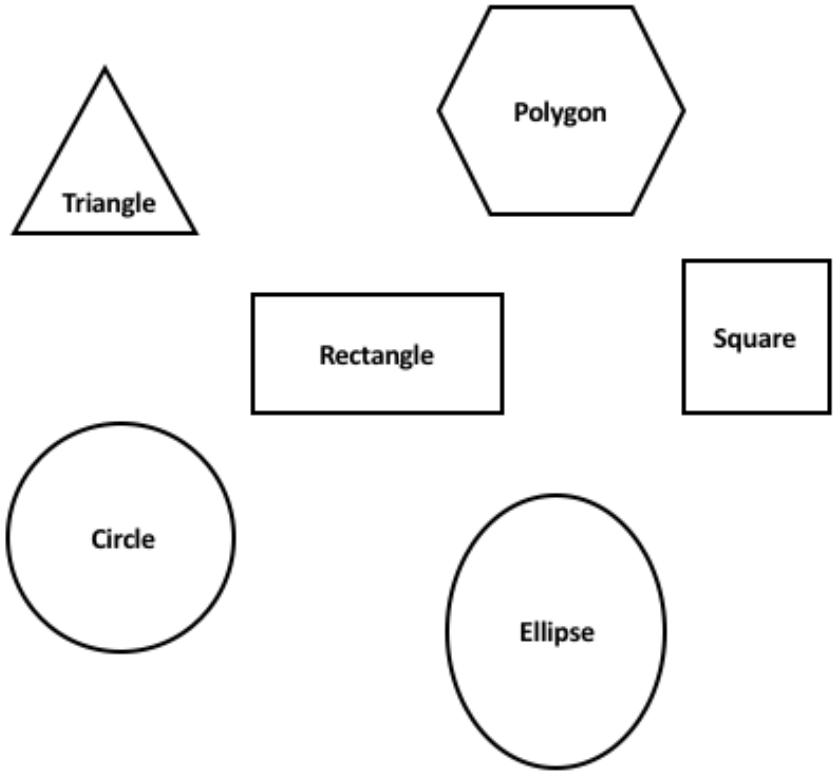
A simple example.

Geometrical Shapes

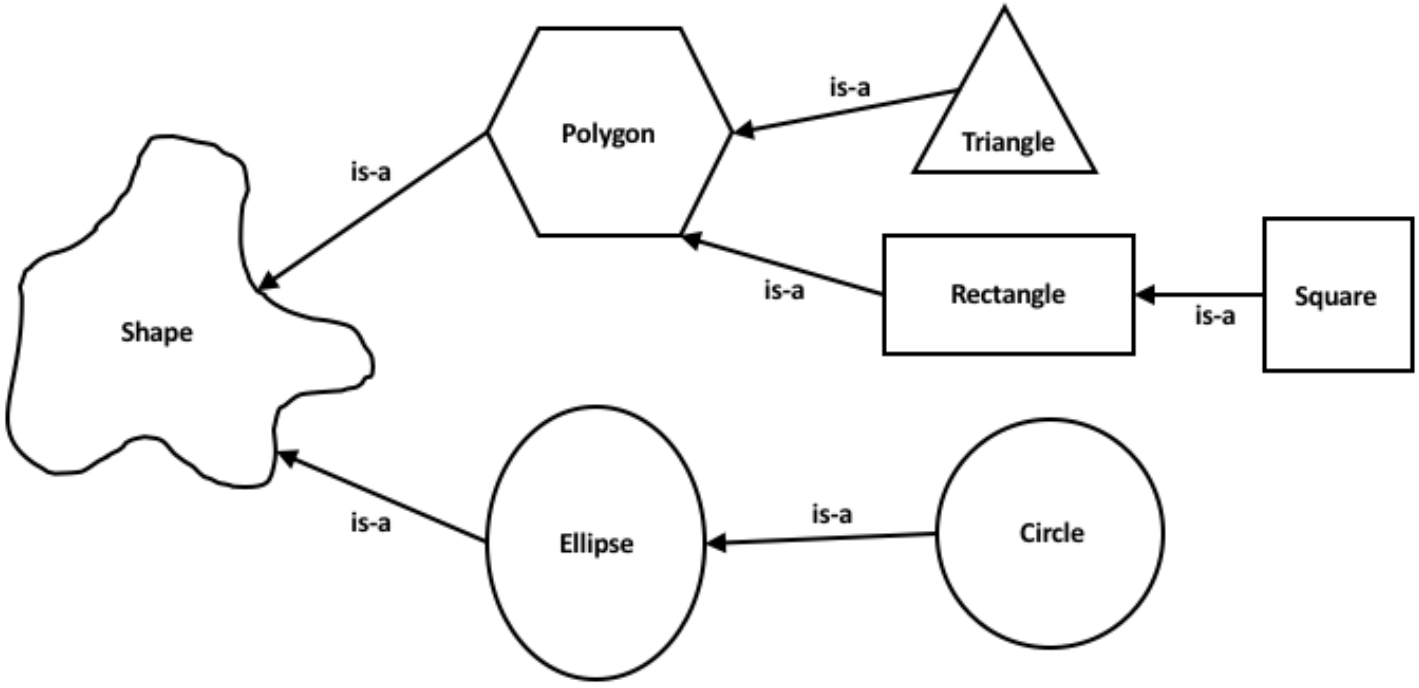
Consider various geometric shapes.

You want to write a program to, among other things, compute the area and perimeter of each shape.

How do we go about it?



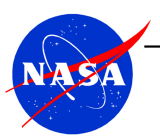
Abstraction and Inheritance



Encapsulation (1)

Encapsulation is the ability to isolate and hide implementation details within a software subsystem.[1]

- Encapsulation allows the creation of an object
- It is the mechanism can shield from outside interference or misuse.
  - Within an object, *some* of the code and/or data may be private to the object and inaccessible to anything outside the object. In Fortran we use the keyword **abstract**.

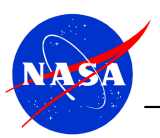


[1] Note: Fortran 90 introduced strong encapsulation capabilities with public/private access for module entities.

Encapsulation (2)

The Fortran concepts for encapsulation are *derived types* and *type bound procedures*[2]

```
1  module my_mod
2     implicit none
3     private ! hide everything by default
4     public my_type ! expose my_type
5     type my_type ! a derived type
6        private ! hide data details
7        real :: value
8     contains
9        procedure :: compute ! a public type-bound procedure
10    end type my_type
11 end my_mod
```

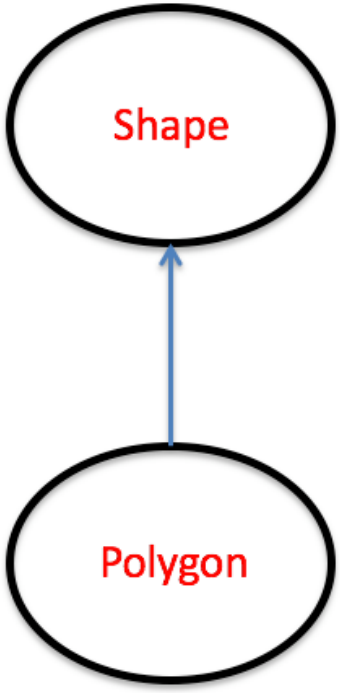


[2]Introduced in Fortran 2003

SSAI

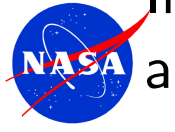
Data Abstraction

Abstraction can be thought of as a natural extension of encapsulation.



- Abstraction refers to the act of representing essential features without including the background details or explanations.
- Classes use concept of abstraction and are abstract data types
- E.g. polygon is an abstraction of shape. Shape is a generalization of polygon.
- Fortran 2003 supports data abstraction (keyword: **abstract**)

Implementation changes, e.g. a software update, rarely affect the abstraction you use.



Inheritance

Inheritance is a way to form new classes using classes that have already been defined.

- Original class is referred to as the **base** class (or **parent** class)
- New class is referred to as the **child** class or **subclass**
- Intent is to reuse significant portions of base class.
- Inheritance relations always form hierarchical trees.
- Fortran 2003 introduces inheritance (keyword: **extends**)
- Child class should be usable in any context where the base class is usable.
  - Useful notion: "is-a" relationship categorization:
    - frog is-a kind of amphibian
    - sparse-matrix is-a kind of matrix
    - polygon is-a kind of shape

Polymorphism

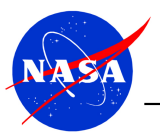
Polymorphism is the capability of treating objects of a subclass as though they were members of the parent class.

- A **polymorphic variable** is one whose actual type is not known at compile time.
  - Run-time environment calls the appropriate methods on depending on actual type (or **dynamic** type)
  - Implemented with **dynamic binding** (usually function pointers)
- Polymorphism and inheritance are distinct aspects but are typically applied together for maximum impact.
- E.g. polymorphic variable *my_shape* of *class shape* will compute the compute area/perimeter according to type set at run time.

Templates

AKA **Parametric Polymorphism**.

- Some languages support the ability to declare multiple similar classes simultaneously.
  - Routines using the type then specify which case to use.
- Fortran 2003 introduces a limited form[3].
  - Derived types can be parameterized for *kinds* and sizes.
  - Cannot parameterize integers and reals simultaneously.

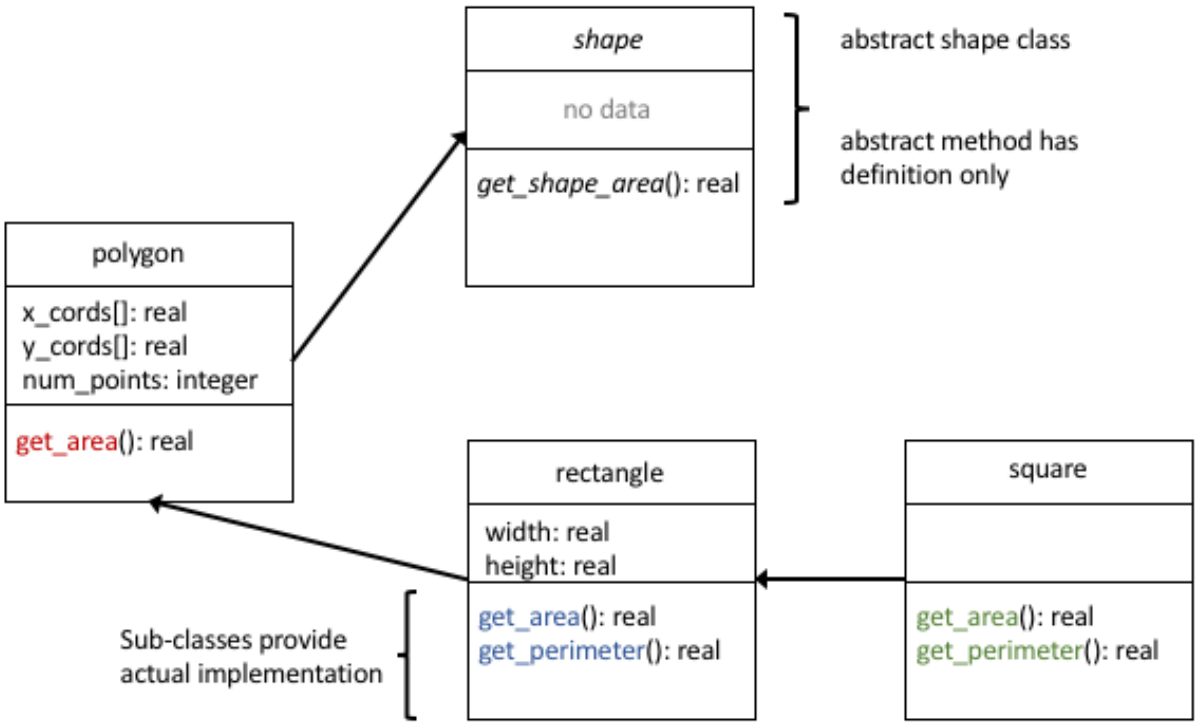


[3] Parameterized derived types

SSAI

Design

UML diagram

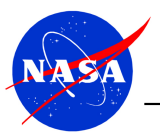


Complex examples.

OOP and Model Infrastructure

The clearest case for OOP in scientific models is in the "infrastructure" which manages the various model abstractions.

- Infrastructure includes
  - I/O
  - Computational grid
  - Loop constructs
  - Domain decomposition
  - Calendars/clocks
- Common infrastructure issues among various Earth system models led to the creation of the ESMF[4]. While not truly OO, ESMF is strongly encapsulated and has an object based look-and-feel.



[4]Earth System Modeling Framework

SSAI

Other examples with significant use of Fortran 2003

- NASA GISS modelE tracer infrastructure
  - Supports **multiple** tracer/chemistry groups
  - Needs to support multiple integration schemes
- The parallel Fortran logging framework for HPC applications
  - Supports multiple **logging** levels and multiple **output** streams
- pFUnit: A parallel Fortran unit testing framework for HPC applications

Conclusion

- F2003 includes a solid support of object orientation.
- Provides opportunities to adopt newer technologies and modernize current earth science models.
- There is already a F2008 standard, but enhancements are "minor" (submodules, co-arrays).

References:

- John Reid, "The new features of Fortran 2003, ACM SIGPLAN Fortran Forum 96", 10 (2007)
- http://www.nag.com/nagware/NP/doc/nag_f2003.pdf
- http://www.pgroup.com/doc/pgifortref.pdf