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.

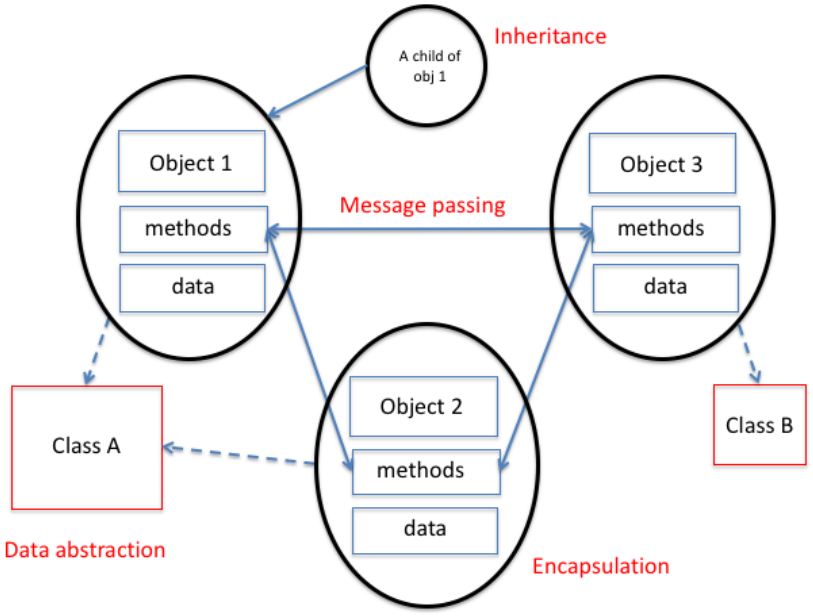
encapsulation
inheritance

- 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.

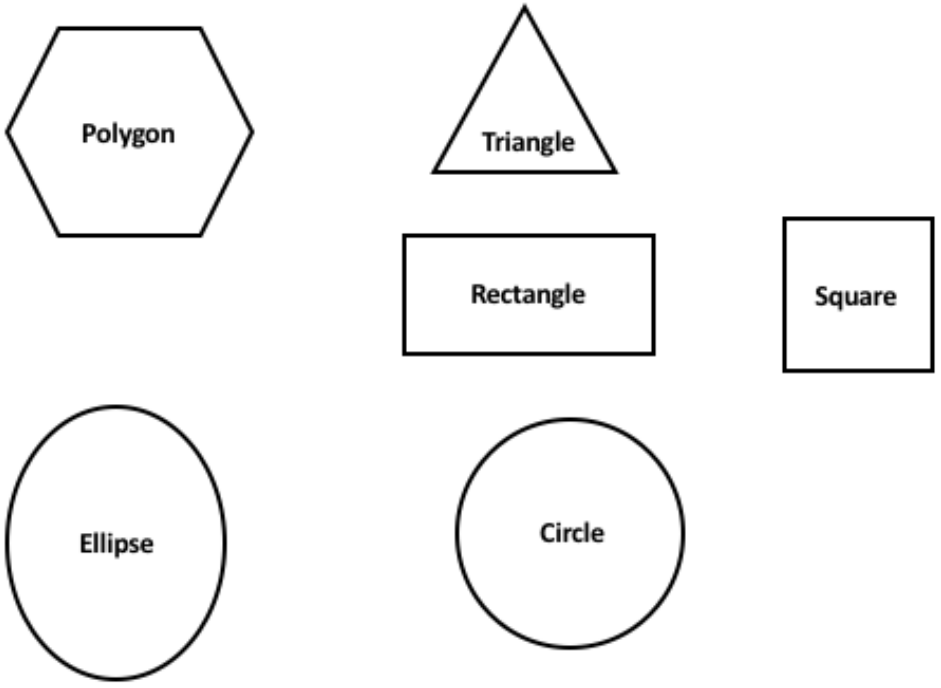
Within a program, objects interact with each other by sending messages (i.e. invoking methods)

How does Fortran 2003+ support OOP?

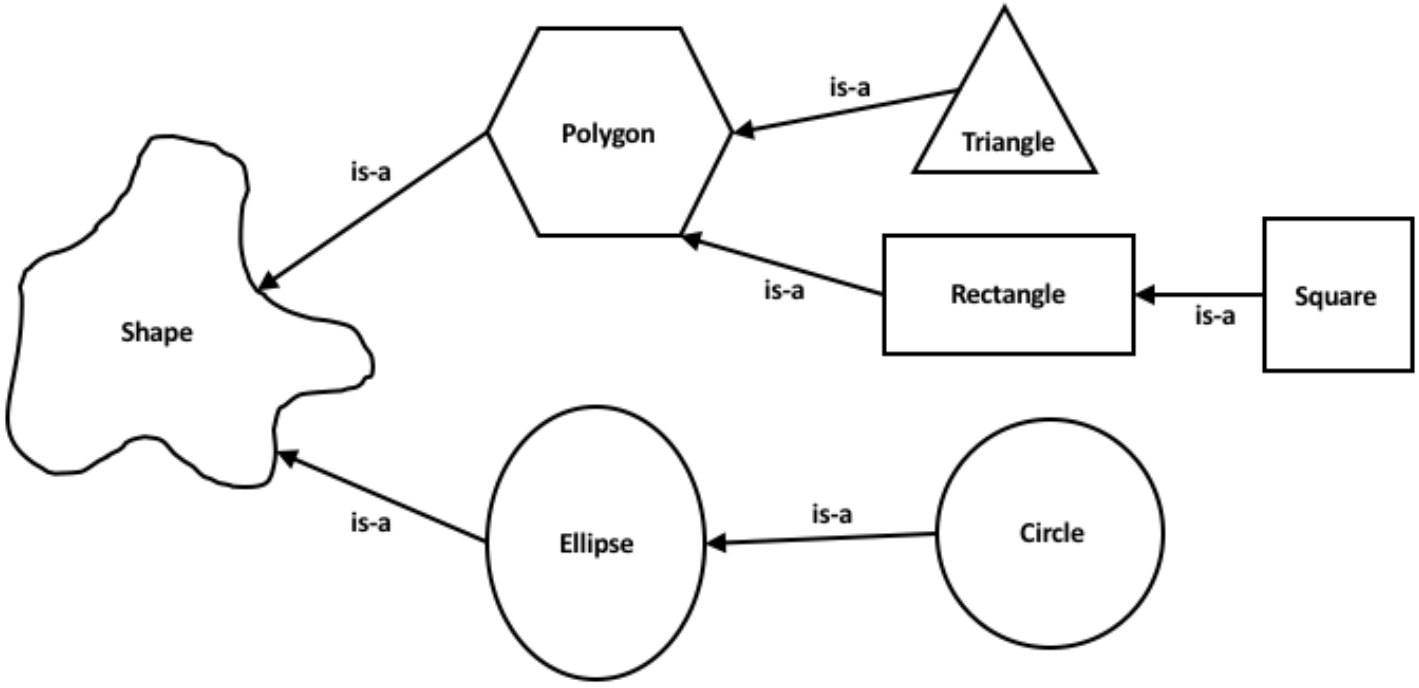
Geometrical Shapes

Consider various geometric shapes.

You want to write a program to compute the area and perimeter of each shape.

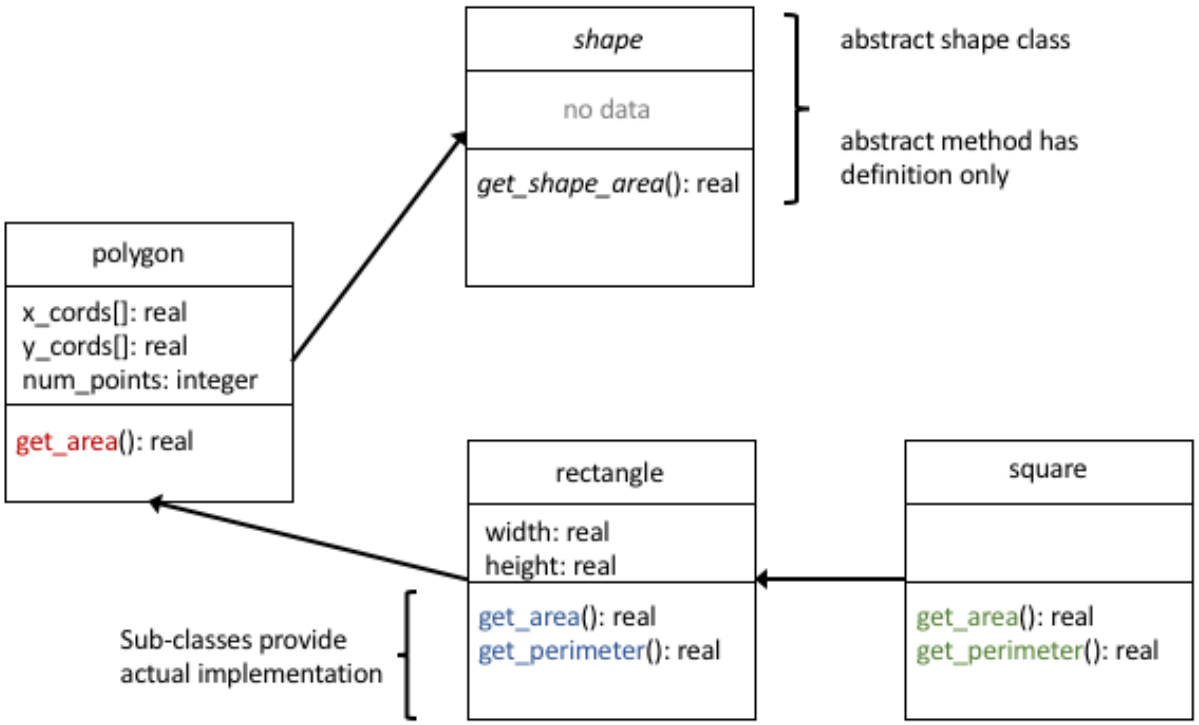


Abstraction and Inheritance

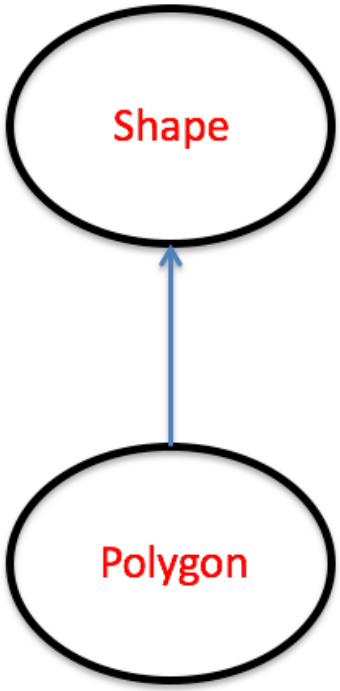


Design

UML diagram



Data Abstraction

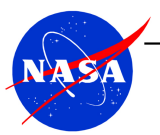


- Abstraction refers to the act of representing essential features without including the background details or explanations.
- Classes use concept of abstraction and are abstract data types
- E.g. polygon is an abstraction of shape. Shape is a generalization of polygon.
- Fortran 2003 supports data abstraction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

Encapsulation

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.

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

Function/procedure pointers

```
1       type :: some_type
2           real :: some_var
3       contains
4           procedure :: some_proc
5       end type some_type
```

While not strictly an OO concept, function pointers (e.g. *procedure :: some_proc*) are a major part of the implementation of OO abstractions. **More on this later**.

- A function pointer is a *data type* that is able to be associated with actual functions/procedures. The association is determined at *run-time*.
- Data structure with function pointer can be used to invoke different behavior in different contexts by associating with different actual functions.
- Introduced in Fortran 2003

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 *myShape* of *class Shape* will compute the compute area/perimeter according to type set at run time.

Templates

AKA **Parametric Polymorphism**.

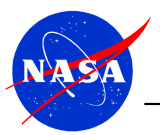
- Some languages support the ability to declare multiple similar classes simultaneously.
  - Routines using the type then specify which case to use.
- Fortran 2003 introduces a limited form.
  - Derived types can be parameterized for *kinds* and sizes.
  - Cannot parameterize integers and reals simultaneously.

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[2]. While not truly OO, ESMF is strongly encapsulated and has an object based look-and-feel.



[2]Earth System Modeling Framework

Other examples

- Climate model tracer infrastructure
  - Needs to support multiple tracer/chemistry
  - Needs to support multiple integration schemes
- Multiple Computational Grids
  - E.g. for coupled Earth systems we might have
    - Lat-Lon (Arakawa A, B, C, D)
    - Cubed-Sphere
    - Icosahedral
  - Some subsystems can "work" with any grid, while others are dependent on specific representations.
  - Coupling can require custom interpolations between grids.
  - Can we provide a software layer that supports various grid-specific operations while hiding the details from the layers that don't really care which grid is being used?

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