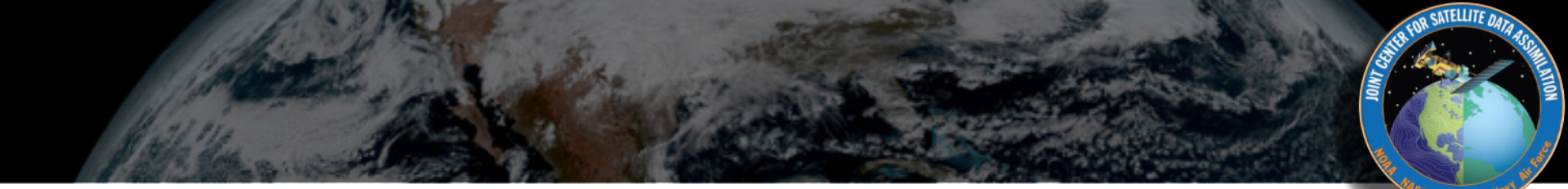


The Joint Effort for Data assimilation Integration

Object Oriented Programming

Joint Center for Satellite Data Assimilation (JCSDA)

Objective

Very brief introduction to object oriented programming

Concepts that are the most useful to work with JEDI

Learning all the subtleties of OOP can take years

What is an object?

An object is an instance of a class, it contains data and methods

The data part of a class is most of the time private (ensure encapsulation)

Methods are instructions sent to objects of a class to perform actions (often using the data)

Methods can be public or private (or protected)

The interface stays, the implementation can change

Polymorphism and inheritance

Subclasses can be defined from a base class

- A subclass inherits the data and methods from its parent class
- A subclass can add its own data and methods
- A subclass can overwrite (specialize) an inherited method

Base classes are very useful for

- Factorizing common code between subclasses
- Defining common interfaces (abstract base class)

It is very common to hold objects whose subclass is not known in pointers to the base class

- Use smart pointers to avoid memory leaks
- Use factories to create objects based on input parameters

Each object carries its methods so the appropriate one will be called

Inheritance warnings

Deep inheritance structures are usually not a good idea

- There are often simpler ways to solve the problem (composition)
- It is not efficient (resolving of virtual tables)
- They introduce too much “coupling”

The deepest inheritance tree in OOPS (Minimizer) has two levels

- First level sets the RHS (primal, dual, saddle point)
- Second level to select the actual minimization algorithm

An object of a subclass can be passed **anywhere** an object from a parent class is expected (there is **no** exception to this rule)

If exceptions (if statements to determine the subclass) are needed, inheritance is not the right approach

Inheritance example


```
class CovarianceMatrix {
 public:
  CovarianceMatrix(const Geometry &, const Configuration &);
  ~CovarianceMatrix();
  virtual Increment multiply(const Increment &) const =0;
}

class SpectralCovariance : public CovarianceMatrix {
 public:
  CovarianceMatrix(const Geometry &, const Configuration &);
  ~CovarianceMatrix();
  Increment multiply(const Increment &) const;
}

class WaveletCovariance : public CovarianceMatrix {
 public:
  CovarianceMatrix(const Geometry &, const Configuration &);
  ~CovarianceMatrix();
  Increment multiply(const Increment &) const;
}
```

Inheritance example


```
class State {}

class OceanState : public State {}

class AtmosphereState : public State {}
```

```
class Model {
  void forecast(State &, const Duration) const =0;
}
```

```
class OceanModel : public Model {
  void forecast(State &, const Duration) const;
}

class AtmosphereModel : public Model {
  void forecast(State &, const Duration) const;
}
```

Inheritance is not for everything!

Generic Programming

```
module generic_min
  interface min
    module procedure min_int, min_real
  end interface

contains

  function min_int(x, y) result (z)
    integer x, y, z
    if (x<y) then
      z = x
    else
      z = y
    endif
  end function min_int

  function min_real(x, y) result (z)
    real x, y, z
    if (x<y) then
      z = y
    else
      z = x
    endif
  end function min_real
end module generic_min
```

It is possible to write generic subroutines or functions in Fortran

However, the code is repeated for each implementation

In C++ the code is not repeated:

```
template<class T>
T min(const T x, const T y) {
  if (x < y) {
    return x;
  } else {
    return y;
  }
}
```

The advantage becomes obvious when the function is long and complex: 4D-Var cost function…

Generic Programming

Templates in C++, “duck-typing” in python, nothing in Fortran

OOPS/JEDI uses generic programming more than inheritance

The technique of TRAITS is used to define a coherent set of classes:

```
struct QgTraits {
  typedef qg::GeometryQG                Geometry;
  typedef qg::StateQG                   State;
  typedef qg::ModelQG                   Model;
  typedef qg::IncrementQG               Increment;
  typedef qg::ErrorCovarianceQG         Covariance;
// this example is incomplete
}
```

Actual classes

Names used in JEDI

General Comments

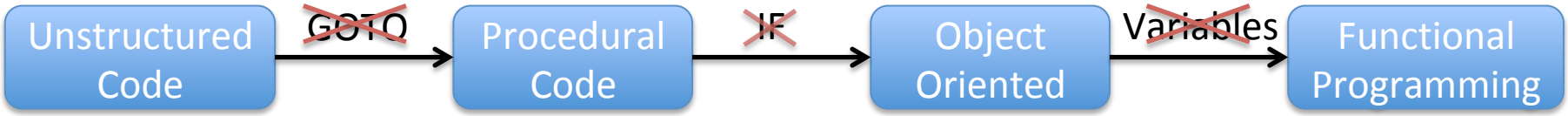
One class = One responsibility

The purpose of a class should fit in one (simple) sentence

Interfaces should be easy to use correctly, difficult to use incorrectly

They should make sense for somebody who knows the domain

Evolution of Programming



Each new programming model removes the most common source of bugs in the previous one