Introduction to Object- and Component-Oriented Programming

Craig Rasmussen

Advanced Computing Laboratory
Los Alamos National Laboratory

Los Alamos NATIONAL LABORATORY

1

Why Use Components?

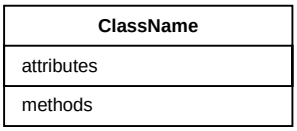
Hero programmer producing single-purpose, monolithic, tightly-coupled parallel codes

- Promote software reuse
  - "The best software is code you don't have to write" *[Steve Jobs]*
- Reuse, through cost amortization, allows
  - thoroughly tested code
  - highly optimized code
  - developer team specialization

2

An Object Is …

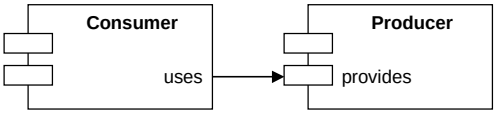
- A software black box
- Contains data
  - data is normally hidden from users
  - "tall fences make good neighbors"
- Provides information based on its data
  - via a method (function)
- A class is software (code) defining an object
  - many object instances can be created from one class



3

A CCA Component Is …

- A software black box
- Contains data
  - uses port
    - specifies required connections to other components
    - function caller
  - plus other data
- Provides information based on its data
  - provides port
    - function callee



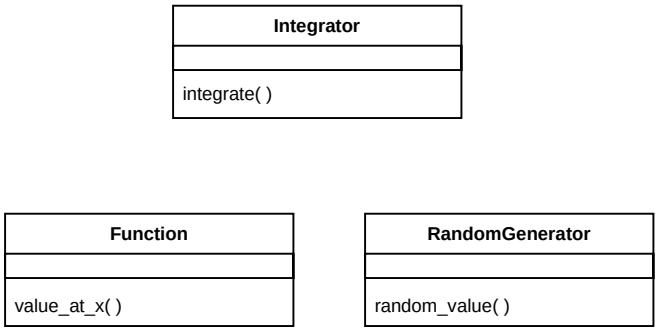
4

What's the Difference?

- More similar than different
  - Objects +
- Components can discover information about the environment
  - from framework
  - from connected components
- Interface discovery
  - find provides ports
  - find uses ports
  - insure that connection can be made between two components
- Easily convert from an object orientation to a component orientation
  - automatic tools can help with conversion

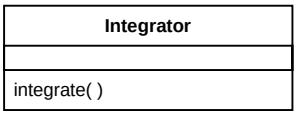
5

Classes



6

Interface Declaration



Integrator.h

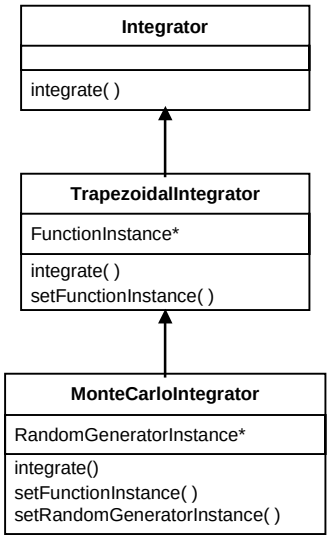
```
class Integrator
{
   virtual void integrate( ) = 0;
};
```

Integrator.f90

```
MODULE IntegratorModule
  interface integrate
    module procedure
random_integrate
  end interface
END MODULE
```

7

Inheritance



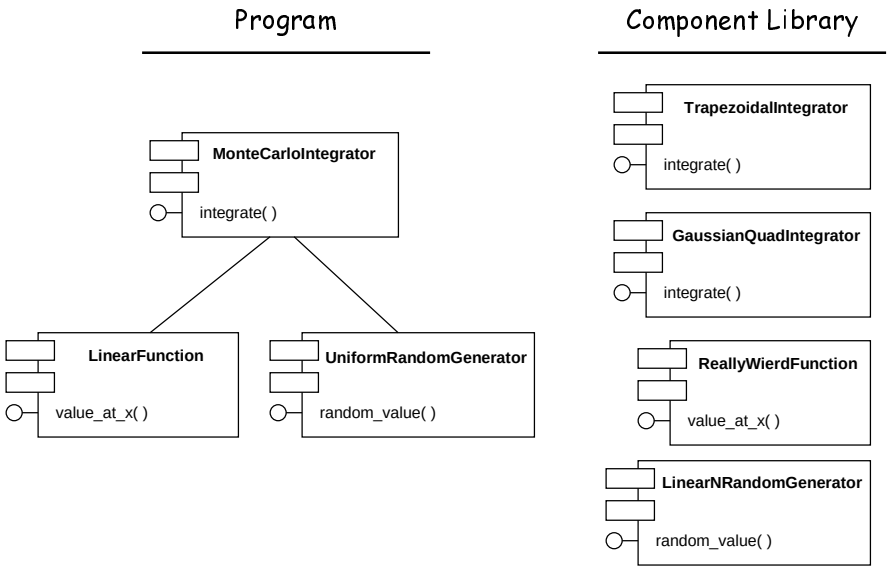
8

Object-Oriented Program

```
#include <iostream>
#include "Function.h"
#include "Integrator.h"
#include "RandomGenerator.h"

int main(int argc, char* argv[])
{
  LinearFunction* function = new LinearFunction();
  UniformRandomGenerator* random = new UniformRandomGenerator();
  MonteCarloIntegrator* integrator = new MonteCarloIntegrator();

  integrator->setFunction(function);
  integrator->setRandomGenerator(random);

  cout << "Integral = " << integrator->integrate(100) << endl;

  return 0;
}
```

9

Component Program



10

Demonstration

11