



OOP for n00bz!

An Introduction to Object-Oriented
Programming in ColdFusion

Matt Woodward

cf.Objective() 2006



About Me

- Technical Architect and Lead Developer for Realty InfoLinks in Dallas, Texas
- ColdFusion Developer since 1996
- Adobe Community Expert for ColdFusion (formerly Team Macromedia)
- Also develop in Java, Flex, PHP, and a bit of C# on occasion

Agenda

- What's an object?
- What's object-oriented programming (OOP)?
 - Contrasts with procedural programming (i.e. CF 5 and earlier)
 - Why use it?
- Fundamental Concepts of OOP
- What makes OOP so great?

What's an Object?

- In ColdFusion when we discuss objects, we're talking about ColdFusion Components (CFCs)
- At their simplest, objects can be collections of functions
 - Can use CFCs to remove business logic from the presentation layer, but this isn't really OOP
- At their most powerful, objects are software models of real-world objects
 - For example, a Person, an Account, or a Shopping Cart

What's an Object?

- Software objects that model real-world objects contain both the data (attributes) related to that object as well as the methods that operate on that data (behavior)
 - This is a sharp distinction from procedural programming, in which data and methods that manipulate the data are separate
- This bundling together of attributes and behavior is called *encapsulation*
 - Makes software safer, easier to debug
 - No concept of “global” variables

Sample Object

- Person
- Has attributes and behavior
- Attributes (data)
 - Name, birthday, sex, height, weight, eye color
- Behavior (methods)
 - Eat, sleep, walk, write ColdFusion code

A Touch of Class

- In OO terms a *class* is a generic object
 - A Person object that hasn't been *instantiated*, i.e. assigned its attributes such as name, etc., is a class
 - Think of a class as a cookie cutter from which you will form specific objects; these are your generic CFCs
- An *object* is really a particular instance of a class

The Person Object

- Very simple Person CFC
- Attributes
 - First name, last name
- Methods
 - `init()`, a.k.a. the “constructor”
 - Getters and setters for each attribute (`get/setFirstName()`, `get/setLastName()`)

What's Object-Oriented Programming (OOP)?

- OOP is a way of creating a software application from objects
- The application itself is made up of messages that are passed between the objects in the system
- OOP contrasts with *procedural programming*

Procedural Programming

- “CF 5” style programming
 - Lots of mixture of layers of the application, e.g. queries and output of these queries on the same CFML page
- Data and methods that operate on the data are separate from one another
- With data out in the open (unprotected), can make for dangerous situations
 - Can also make debugging difficult
- As applications grow in size, the procedural model begins to break down and become unmanageable

OOP

- Data and methods are contained within objects
 - Data is protected
 - Application can grow with fewer problems, easier debugging
- Objects model real-world constructs
 - Conceptually even complex problems are simpler to address through software modeling

Thinking in OO

- MOST IMPORTANTLY: ***OO is a way of thinking***
- OO itself isn't particularly difficult; ***change***, however, often is
- The vast majority of concepts in this presentation are not CF-specific
 - I first learned OO concepts as a Java developer, and these concepts transfer directly over into CF development

Just Do It

- Everything's an object! (Java concept)
- Be aware of tell-tale signs you're not thinking in OO
 - Data hanging out for all the world to see
 - Too much business logic on your presentation pages
 - Lack of real-world object "nouns" in your application
 - Not many CFCs in your application
- Don't be afraid to get your hands dirty!

OOP in ColdFusion

- CF's "object" is the CFC
- CFCs are more than UDF collections!
- CFCs are OO
 - Many rather esoteric debates about how OO CFCs are
 - Don't worry about this—CFCs are OO and are extremely capable in this role
 - In many cases CFCs are more flexible than Java objects! Dynamic typing can be a blessing, not a curse.

Why OOP?

- Everybody's doing it—and with good reason!
 - Java, C#, Ruby, PHP, ActionScript, C++, etc. are all OO languages
- Been around since the 1960s, just “new” to CF
 - Not that new to CF anymore!
- Understanding OO concepts will help your career in addition to your applications
- OOP is an extremely proven method of building software
 - It's not a debate anymore!

Why OOP?

- It will make your life easier
 - Easier maintenance, easier debugging, better data protection, simplifies complex applications, better code reuse, etc. etc. etc.
- No more excuses!
 - We're well into the second generation of CFCs in CF—if you're not using them, why not?

Contact Manager Application

- Very simple example application
- Two versions: one procedural, one OO
- Illustrates the differences in an application we've all built numerous times
- Small number of objects, but the concepts are identical to those you would use in a large application
 - Large applications just have more objects!

Object #1: Person.cfc

- Similar to the Person CFC we looked at earlier, just a few more attributes
- The “nouns” in your application should model real-world objects
- Person.cfc has attributes (data) and methods that manipulate the data all inside the object itself
- Specifically this is a “bean” (Java term)
 - Simple pattern—object with attributes and getters and setters for the attributes

Scopes: *variables vs. this*

- The *this* scope is not private to the CFC
 - Can be manipulated directly by outside processes
 - Does *not* correspond to the *this* scope in Java!
- The *variables* scope is private to the CFC
 - We create our objects in such a way that to get or set the data, the outside process has to ask
- Short answer: don't use the *this* scope!

Scopes: “var” Scope Everything!

- Always, always, **always** use the var scope within your methods
 - If you don't the variable will “leak” out of the function and cause unwanted side-effects and threading issues
- Must do this for explicitly created variables (don't forget things like loop counters!) AND for ANYTHING that will CREATE variables (e.g. queries, cfhttp calls, etc.)
- This point can't be overstated—if you don't do this, weird stuff will happen!

Other Scope Considerations

- CFCs share the form, URL, request, CGI, cookie, client, session, application, server, and Flash scopes with the calling page
- You usually don't want to use these directly even if they're available
 - Reduces code reuse due to tight coupling

Cohesion and Coupling

- Cohesion and coupling are extremely important concepts to the OO thought process
- Cohesion: the degree to which an object is singular in purpose
 - Tight cohesion is good, loose cohesion is bad
- Coupling: the degree to which one object is dependent upon another
 - Loose coupling is good, tight coupling is bad
- The goal: *tight* cohesion and *loose* coupling

Make Me a Better Person

- Updated version of Person object
- Exact same concepts, just some additional attributes and methods
- Still a “bean” type object
- Adds a `validate()` method to check the data as well as a private `isEmail()` method

The Object-Relational Mismatch

- Our application is built with objects, but our database is (likely) relational
- Don't build your objects based on your database tables or vice-versa
 - Build your database last!
- RDBMS are good at what they do and objects are good at what they do
 - A translation between the two needs to occur
 - We're going to do this manually with a Data Access Object (DAO)
 - Frameworks like Reactor are designed to handle this automatically

Object #2: PersonDAO.cfc

- Data Access Objects (DAOs) handle persistence of objects into the application's data repository
- DAOs deal with the “bean” objects and handle Create, Read, Update, and Delete operations (“CRUD”)
- DAOs abstract the specifics of persistence from the rest of the application

Object #3: PersonGateway.cfc

- Gateway objects deal with multiple records in the data repository, e.g. `getAllPeople()`
 - DAOs are for *single* records
 - Gateway methods often return a query object
- Like DAOs, gateways serve the purpose of abstracting the specifics of persistence from the rest of the application
 - Loose coupling in action



Contact Manager Objects

- BetterPerson.cfc: models a real-world person
- PersonDAO.cfc: handles persistence for the BetterPerson bean
- PersonGateway.cfc: handles multiple records in the application

Using CFCs

- `<cfobject>` or `CreateObject()`
 - Creates an instance of or “instantiates” the CFC
- `<cfinvoke>`
 - Creates the object, calls the method specified, and destroys the object (*transient* objects)
 - Don't use `<cfinvoke>` if you're going to make multiple calls on an object

Talk Is Cheap

- Time for some code!
 - Walk-through of application
 - Comparison of procedural and OO functionality

OO Goal #1: Tight Cohesion

- Do one thing and (hopefully) do it well
- Many small objects are better than large monolithic ones
- Tight cohesion makes for easier maintenance, easier debugging, and easier reuse of objects

OO Goal #2: Loose Coupling

- Reduce or eliminate dependencies between objects
- Objects should be self-contained and should need to know little to nothing about other objects in the system
- Easier maintenance, easier debugging, easier code reuse

OO Advantages

- Encapsulation
 - Data and methods in self-contained objects
- Code reuse
 - Tight cohesion, loose coupling
- Better, more flexible code organization
- “Low impact” maintenance
- Simplifies complex applications
 - Simpler conceptual architecture, easier coding
- Less “fragile” applications
 - At least this was my visceral reaction when first learning OO!

For You OO Junkies ...

- CFCs allow for:

- Inheritance
- Use of super (was new in CF 6.1)
- Introspection (cfproperty, getMetaData())
- Composition/Aggregation (CFC inside another CFC)
- Method overriding
- Polymorphism

- Lacking are:

- Interfaces (who needs 'em?)
- Method overloading
- True constructors (we use init())

Summary

- OOP is a tried-and-true way to build software
- De facto standard supported by every major language/technology
- Offers simple, elegant solutions to complex problems
- Allows us to build scalable, secure, large applications more easily
- **JUST DO IT!**

Online Resources

- <http://www.halhelms.com>
(esp. newsletters)
- <http://www.mattwoodward.com/blog>
- <http://www.corfield.org/blog>
- <http://www.clearsoftware.net>
- <http://www.doughughes.net>
- <http://java.sun.com/developer/onlineTraining/new2java/index.html>

Books

- Weisfeld, Matt. *The Object-Oriented Thought Process*. 2d ed. Sam's Publishing, 2004.
- Sierra, Kathy and Bert Bates. *Head First Java*. O'Reilly, 2004.
- Eckel, Bruce. *Thinking in Java*. Available online: <http://www.thinkinginjava.com>
- Taylor, David A. *Object Technology: A Manager's Guide*. 2d ed. Addison-Wesley, 1998.

Questions?

- E-Mail Me: mpwoodward@gmail.com
- Skype the ColdFusion Weekly Podcast!
 - Skype User: cfweekly
 - Phone: 469-293-3820
 - Come record a question here and we'll put it on the podcast!