

# Heresy! Embracing Duck Typing in CFCs

Sean Corfield  
Chief Systems Architect  
Broadchoice, Inc.

# What is this about?

- It's a simple concept hiding behind a stupid phrase!
- Michael Dinowitz  
(Fusion Authority  
Quarterly Update  
Summer 2006)



# What is this about?



- ColdFusion is a dynamically typed language - that's a good thing - but we can't leverage that if we keep writing CF code like Java code
- Who needs types?

# Who am I?

- Chief Systems Architect @ Broadchoice
- Adobe Community Expert for ColdFusion
- Formerly Senior Architect @ Macromedia
- ColdFusion developer since late 2001
- Java developer since early 1992
- Occasional Ruby, Groovy, Smalltalk developer

# What is a type?

- We'll start with built-in types...
- Numeric, String, Date etc
  - Think "cfparam"
  - Specifies permitted range of values

# What is a type?

- Specifies permitted operations:

- `x = "one" ; y = "2" ; z = 3;`

- `a = x * y; // illegal, x not numeric`

- `a = y * z; // OK, a = 6`

- `a = x & y; // OK, a = "one2"`

- `a = y & z; // OK, a = "23"`

- `a = a * 2; // OK, a = 46`

# What is a type?

- At **runtime**, ColdFusion checks whether a value is, or can be converted to, the required type
- An exception is thrown if the conversion is not possible
- **Values have type** (remember this!)

# Working with types

- Tags that work with "type":
  - `cfparam (type=)`
    - (yes, there are several functions too)
  - `cfargument (type=)`
  - `cffunction (returntype=)`
  - `cfinvoke (component=)`
  - `cfobject (component=)`

# A sea change in ColdFusion

- Prior to ColdFusion MX, very little type checking - just validation of user data
- ColdFusion MX introduced user-defined types (components) and much stronger notion of what is a type - apparently

# What about components?

- User-defined type, e.g., Student
- Permitted values:
  - Any object (value) of type Student
  - Any object of a type that extends Student
- Permitted operations:
  - Any method (function) declared in Student or any type that Student extends

# Permitted component values

```
<cfcomponent> <!--- Course --->  
  <cffunction name="enroll">  
    <cfargument name="s" type="Student" />  
    ...
```

```
cs101 = createObject("component", "Course");  
andy = createObject("component", "Student");  
jane = createObject("component", "Sophomore");
```

```
cs101.enroll(andy);  
cs101.enroll(jane);  
// OK, assuming Sophomore extends Student
```

# Permitted component operations

```
<cfcomponent extends="Person"> <!-- Student --->
```

```
...
```

```
andy = createObject("component", "Student");  
andy.study(); // Student.study()  
andy.takeTest(); // Student.takeTest()  
years = andy.getAge(); // Person.getAge()
```

# Permitted component operations

```
<cfcomponent extends="Person"> <!-- Student -->
```

```
...
```

```
andy.study();           // Student.study()  
andy.takeTest();        // Student.takeTest()  
years = andy.getAge();  // Person.getAge()
```

- ColdFusion doesn't care what type andy is - only whether the methods exist!
  - andy.fly(); // crash'n'burn!

# Inspired by Java?

- With CFCs, we went OO
- Since CFMX was built on Java, we looked to Java for guidance with OO
- Java is a strongly typed language so we wrote return types and argument types for all our methods - just like Java!

# ColdFusion != Java

- In Java we declare variables with types:
  - `String x = "I am a string";`
  - `int y = 2;`
  - `Student dave = new Student();`
  - `x = 42; // nope, x is not int`
  - `y = "42" ; // nope, y is not String`
- Variables have type (unlike ColdFusion)

# Dave goes flying?

- `dave.study();`
- `dave.takeTest();`
- `dave.fly(); // will not compile!`
- Java prevents this mistake

# What if he never really flies?

- `if (false) dave.fly();`
- Java still won't compile this
- Student does not have a `fly()` method

# Dave goes to ColdFusion classes

- `<cfargument name="dave" type="Student" required="true" />`
- `<cfset dave.study() />`
- `<cfset dave.takeTest() />`
- `<cfif false>`
  - `<cfset dave.fly() />`
- `</cfif>`

# One hand clapping?

- If a method does not exist in an object and no one is there to call it, does it still throw an exception?
- `<cfif false>`
  - `<cfset dave.fly() />`
- `</cfif>`

# Does ColdFusion care about Dave?

- `<cfargument name="dave" type="Student" required="true" />`
- Will check that a value is passed in and that it is an object of type Student (or something that extends Student) - at runtime

# Does ColdFusion care about Dave?

- `<cfargument name="dave" type="Student" required="true" />`
- `<cfset dave.study() />`
- Will check that the value passed in has a method called `study()` - at runtime

# Does ColdFusion care about Dave?

- `<cfargument name="dave" type="Student" required="true" />`
- `<cfset dave.study() />`
- `<cfset dave.takeTest() />`
- Will check that the value passed in has a method called `takeTest()` - at runtime

# Dave who?

- `<cfargument name="dave" type="any" required="true" />`
- `<cfset dave.study() />`
- Will still check that a value passed is passed in and that it has a method called `study()` - at runtime (similarly for `takeTest()`)

# Dave who?

- The argument type just determines what type of runtime exception we see if we pass the wrong object in...

# Dave who?

- If he studies like a student and takes tests like a student, he must be a student!

# Enter the Duck

- If it walks like a duck and quacks like a duck, it must be a duck!
- Behavior defines the type, not the other way around



# Programming to an interface

- If a function calls `someMethod()` on an argument, then any object that implements `someMethod()` is acceptable to that function, regardless of that object's actual type
- Behavior is more important than type

# Less Java, More Ruby?

- Java has interfaces - literally - as part of its static type system and the compiler enforces them
- Ruby does not have interfaces - it relies on convention (behavior)
- Smalltalk - the grand daddy of OO languages - also does not have interfaces

# Real-World Example

- Broadchoice authentication for Sections and Pages in the portal application

# Testing, Testing, Testing!

- We have no compiler to catch our silly mistakes - we have to run our code
- If we remove the argument type check, we must be more thorough in our tests
- Unit testing is therefore very important!

# Duck Testing?

- While duck typing makes testing more important, it also helps us write tests
- Duck typing makes it easier to swap real components and mock components
- Mock components do not need to extend the types they stand in for

# Testing scenario - user service

- User service depends on a user DAO
- You want to test this by writing a mock DAO - a component that pretends to read and write to the database (but actually just uses canned, hardcoded data)

# Testing scenario (cont)

- If you specify `type="UserDAO"` then your mock DAO must extend the real DAO in order to be used in the service (this is the Java way)
- Duck typing - `type="any"` - means your mock DAO can be independent of the real DAO - and potentially more reusable (certainly less coupled)

# Testing scenario (cont)

- Brian Kotek's ColdMock project allows you to create mock objects with the exact same type (but still relies on dynamic behavior)
- Even so, mocking code is **much** easier with duck typing (try mocking Transfer!)

# Real-World Example

- Broadchoice unit tests for User and object injector (Brian Kotek's object injector is better!)

# Performance Observation

- ColdFusion performs its type checking (and throws exceptions) at runtime
- If you specify `<cfargument>` types and `<cffunction>` returntypes, then code is executed to perform those tests
- CF8 allows you to disable CFC type checking to improve performance

# Summary

- ColdFusion is a dynamic language that perhaps has more affinity with Smalltalk, Ruby and Groovy than the statically typed Java
- By trying to follow Java's lead too closely we are constraining ourselves

# Next Steps

- Mixins
  - Blending the functionality of two (or more) components without duplicating code
  - "Class-based" using `<cfinclude>`
  - "Object-based" using `structAppend()`

# Resources

- "Vya duck?" - Judith Dinowitz
  - <http://www.fusionauthority.com/Techniques/4588-Vy-a-Duck.htm>
- "Flock the duck!" - Michael Dinowitz
  - Fusion Authority Quarterly Update Summer 2006
- Hal Helms (various):
  - <http://www.halhelms.com/index.cfm?fuseaction=newsletters.show&issue=jan2006>
  - <http://coldfusion.sys-con.com/read/172586.htm>

# Q&A

- [sean@corfield.org](mailto:sean@corfield.org)
- <http://corfield.org/>
- [scorfield@broadchoice.com](mailto:scorfield@broadchoice.com)
- <http://broadchoice.com/>