

# Heresy! Embracing Duck Typing in CFCs

*Sean Corfield*

*Sr Computer Scientist  
Adobe Systems, 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?



- I am not Hal Helms



- Who is that person?



June 28<sup>th</sup> – July 1<sup>st</sup> 2006

# What is a type?

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



June 28<sup>th</sup> – July 1<sup>st</sup> 2006

# 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
```



June 28<sup>th</sup> – July 1<sup>st</sup> 2006

# 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!)



June 28<sup>th</sup> – July 1<sup>st</sup> 2006

# Working with types

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



June 28<sup>th</sup> – July 1<sup>st</sup> 2006

# 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



June 28<sup>th</sup> – July 1<sup>st</sup> 2006

# 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



June 28<sup>th</sup> – July 1<sup>st</sup> 2006

# Permitted component values

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

...

```
andy = createObject("component", "Student");  
cs101.enroll(andy);
```

```
jane = createObject("component",  
  "Sophomore");  
cs101.enroll(jane);
```

OK, assuming Sophomore extends Student



June 28<sup>th</sup> – July 1<sup>st</sup> 2006

# Permitted component operations

```
<cfcomponent name="Student" extends="Person">
```

```
...
```

```
andy.study(); // Student.study()
```

```
andy.takeTest(); // Student.takeTest()
```

```
years = andy.getAge(); // Person.getAge()
```



June 28<sup>th</sup> – July 1<sup>st</sup> 2006

# Permitted component operations

```
<cfcomponent name="Student" extends="Person">
```

```
...
```

```
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!
```



June 28<sup>th</sup> – July 1<sup>st</sup> 2006

# 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!



June 28<sup>th</sup> – July 1<sup>st</sup> 2006

# ColdFusion $\neq$ 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



June 28<sup>th</sup> – July 1<sup>st</sup> 2006

# What if he never really flies?

```
if (false) dave.fly();
```

- Java still won't compile this
  - ✓ Student does not have a fly() method



June 28<sup>th</sup> – July 1<sup>st</sup> 2006

# Dave goes to ColdFusion classes

```
<cfargument name="dave"  
  type="Student" required="true" />
```

```
<cfset dave.study() />
```

```
<cfset dave.takeTest() />
```

```
<cfif false>
```

```
  <cfset dave.fly() />
```

```
</cfif>
```



June 28<sup>th</sup> – July 1<sup>st</sup> 2006

# 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>
```



June 28<sup>th</sup> – July 1<sup>st</sup> 2006

# Does ColdFusion care about Dave?

```
<cfargument name="dave"  
    type="Student" required="true" />
```

- Will check that the value passed in is an object of type Student (or something that extends Student) – at runtime



June 28<sup>th</sup> – July 1<sup>st</sup> 2006

# 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 the value passed in has a method called study() – at runtime (similarly for takeTest())



June 28<sup>th</sup> – July 1<sup>st</sup> 2006

# Dave who?

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



June 28<sup>th</sup> – July 1<sup>st</sup> 2006

# Dave who?

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



June 28<sup>th</sup> – July 1<sup>st</sup> 2006

# 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



June 28<sup>th</sup> – July 1<sup>st</sup> 2006

# 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)



June 28<sup>th</sup> – July 1<sup>st</sup> 2006

# 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!



June 28<sup>th</sup> – July 1<sup>st</sup> 2006

# 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



June 28<sup>th</sup> – July 1<sup>st</sup> 2006

# 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)



June 28<sup>th</sup> – July 1<sup>st</sup> 2006

# Testing scenario (continued)

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



June 28<sup>th</sup> – July 1<sup>st</sup> 2006

# 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



June 28<sup>th</sup> – July 1<sup>st</sup> 2006

# Performance Theory

- If you specify “any” or omit the attributes altogether, it won’t execute code to check the types
- Similarly for the required= attribute
  - ✓ required=“true” executes code to check for the presence of an argument



June 28<sup>th</sup> – July 1<sup>st</sup> 2006

# Performance Reality

- Using duck typing in the core of Fusebox 5 sped up the XML to CFML conversion by a factor of 2-3x!
- Joe Rinehart reported a similar performance boost by adopting duck typing in Model-Glue: Unity!



June 28<sup>th</sup> – July 1<sup>st</sup> 2006

# Summary

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



June 28<sup>th</sup> – July 1<sup>st</sup> 2006

# Next Steps

- Mixins

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



June 28<sup>th</sup> – July 1<sup>st</sup> 2006

# Resources

- “Vy a 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>



June 28<sup>th</sup> – July 1<sup>st</sup> 2006

# Q&A

[sean@corfield.org](mailto:sean@corfield.org)

[Sean.Corfield@adobe.com](mailto:Sean.Corfield@adobe.com)

<http://corfield.org/>



**Better by Adobe.™**



June 28<sup>th</sup> – July 1<sup>st</sup> 2006