Introduction to Object-Oriented Programming with CFML

Matt Woodward

4th-6th June 2008
Edinburgh, Scotland

SCOTCH ON THE ROCKS

About Me

- Principal IT Specialist, Office of the Sergeant At Arms, United States Senate
- CFML developer since 1997
- On-again, off-again Flex and Java developer
- Mach-II Release Coordinator
- Member, Open BlueDragon Steering Committee

Agenda

- Object-oriented programming (OOP) history
- Why OOP emerged
- Problems addressed by OOP
- Advantages of OOP
- Basic OOP principles
- What’s an object?
- The power of objects
- Thinking in objects

OOP History

- First emerged in the 1960s
- Main concept
  - objects contain their own data
  - objects define their own behavior
- First language to contain objects: Simula 67
- First widely used OO language: Smalltalk
  - introduced in 1981 issue of Byte Magazine
  - Dynamic language, not static
- Other early OO languages: Lisp and Pascal

OOP History (cont.)

- OOP became dominant in the mid-1990s
  - C++
  - GUI programming
  - event-driven programming
- OO features have been added to many non-OO languages
- Current popular OO languages
  - Java, C#, VB .NET, Ruby, Python
- ColdFusion introduced CFCs in version 6
  - became much more usable in 6.1
  - big performance improvements in CF 8

Why OOP Emerged

- Software applications ...
  - became more complex
  - became more difficult to develop and maintain
- Procedural model no longer feasible (debatable)
- In OOP, real world objects are modeled in software
  - easier to conceptualize and build larger, more complex applications
- Easier to distribute workload
- Objects contain both data and behavior
  - systems are more stable
  - fewer side effects
  - code is more expressive

Advantages of OOP

- Easier to code
  - software represents real-world objects
- Modular
  - independent components
  - decoupling
- Low-impact maintenance
- Extensible
- Reusable components
  - nice side effect, but don't make this your #1 goal!

What's an Object?

- Represents a real-world object
- Has attributes like a structure
- In addition to attributes also has behavior (methods)
- Simple object example: Person
  - attributes: firstName, lastName, birthdate
  - methods: getFirstName(), getLastName(), getBirthdate(), getAge()

Power of Objects

- Encapsulation and data hiding
  - objects are self-contained
  - objects protect their data and hide implementation details
- Data type safety
  - errors thrown if datatypes not correct
- Data validation
  - can check data before gets/sets

Important Object Design Concepts

- Cohesion
  - “do one thing and do it well”
- Coupling
  - objects should have limited knowledge of one another
- Flexibility
  - easier development
  - easier maintenance
  - applications are not “brittle”

Basic OOP Principles

- Inheritance
- Composition
- Polymorphism
- Encapsulation
- Data Hiding

Inheritance

- “Is a” relationship between objects
- Parent/child relationships
- Child object inherits all attributes and behavior from parent objects
  - can also add its own functionality
  - example: person “is an” animal

Composition

- “Has a” relationship between objects
- Objects can contain other objects
- Example: Person “has an” address
- “Favor composition over inheritance” (GoF)

Polymorphism

- “Many shapes”
- Objects can respond to same method calls but behave differently
  - e.g. draw() for a circle vs. draw() for a square
- Note that CFML has only method overriding, not method overloading

Encapsulation and Data Hiding

- Related concepts but not the same
- Encapsulation: object’s attributes and behavior are self-contained
- Data hiding
  - data is protected--manipulation only through public methods
  - outside caller does not know how the object is getting its data

OOP in CFML

- A brief detour ...

OOP in CFML

- ColdFusion Components (CFCs) are CFML's version of objects
- Have most of the characteristics of other OO languages
- Properties (attributes) are the private data in CFCs
- Behavior (methods) implemented as UDFs (<cffunction>)

Properties in CFCs

- Always use the variables scope for instance data
  - “this” scope in CFML is NOT the same as “this” scope in other languages
  - my recommendation: never use “this” scope
- <cfproperty> only used for ...
  - documentation
  - web services
  - AS <-> CFC conversion

Methods in CFCs

- Implemented using <cffunction>
- Can include arguments
  - arguments can be required or optional
  - three ways to get arguments into a method
- Methods can either return a single value or not return anything
  - if not returning anything, indicate returntype of void
- You MUST “var” scope all variables local to a method!

Methods in CFCs (cont.)

- Access to methods
  - public: callable from anywhere
  - private: callable only within the CFC
  - package: callable only in the CFCs directory and lower
- Pay attention to pass by value vs. pass by reference
  - value: simple data types and arrays
  - reference: complex data types, structs, and CFCs
- Always set output to false to suppress whitespace

Constructors in CFCs

- Constructor is the method that is called to create an instance of the object
  - in Java: Person person = new Person();
- CFCs do not really have constructors
- Pseudo-constructor area
  - everything between the opening <cfcomponent> tag and the first <cffunction>
  - common practice to create an init() method that returns the object itself

Let’s Dig Into the Details

Encapsulation in Action

- Everything the Person object needs to know (attributes) and do (methods) is contained within the Person object
  - well, almost everything--there is the save() debate ...
- Example: getBirthdate() returns the birthdate
- getAge() calculates the age based on the birthdate
  - caller doesn't know or care how the calculation is being done

Data Hiding in Action

- Data cannot be manipulated directly
  - variables scope vs. “this” scope
  - must call methods to get/set data
- Necessity of calling methods also allows for validation
  - can test for valid date when setting birthdate
  - can implement security on methods, e.g. getSalary()

Encapsulation and Data Hiding Example

- Person.cfc

Inheritance in Action

- Inheritance occurs when an “is a” relationship exists between two objects
- Example: a person “is an” animal
  - Person inherits Animal attributes and behavior
    - eat(), breathe(), sleep()
  - Person also have behavior unique to humans
    - walkUpright(), writeCode()
- Inheritance eliminates redundant code when objects share similar traits

Inheritance in Action

- Inheritance allows objects to ...
  - keep common attributes in a single parent object
  - retain uniqueness
  - be interchangeable via type promotion
- Example: method that accepts an animal object
  - can pass this method a Person object since a Person "is an" Animal
  - if the method only cares about the object being an Animal, could also pass it a Cat object

Inheritance Example

- Shape.cfc, Oval.cfc, and Rectangle.cfc

Composition in Action

- Composition occurs when a “has a” relationship exists between two objects
- Example: a Person “has an” Address
  - two separate objects: Person and Address
  - Person.setAddress() takes an Address object
  - Person.getAddress() returns an Address object
  - Address contains its own attributes and methods
  - Person.getAddress().getCity() would return city
  - Common to use an array of objects for multiple Addresses

Composition vs. Inheritance

- “Favor composition over inheritance” (GoF)
- Single biggest mistake by novice OO programmers is seeing inheritance everywhere
- Inheritance is actually semi-rare
- Not uncommon to have applications that do not have inheritance

Composition Example

- Person.cfc and Address.cfc

Polymorphism in Action

- Object’s ability to respond to the same method call in different ways
  - e.g. Square and Circle are both Shape objects
  - draw() behaves differently for each
- Powerful concept because it’s not necessary to ...
  - Consider an object’s specific type
  - Worry about implementation details
- Another example: employee object might have calculatePay() method
  - type promotion for objects with same parent
  - how pay is calculated differs between salaried and hourly employee

Polymorphism Example

- draw() method in Oval.cfc and Rectangle.cfc

Thinking in Objects

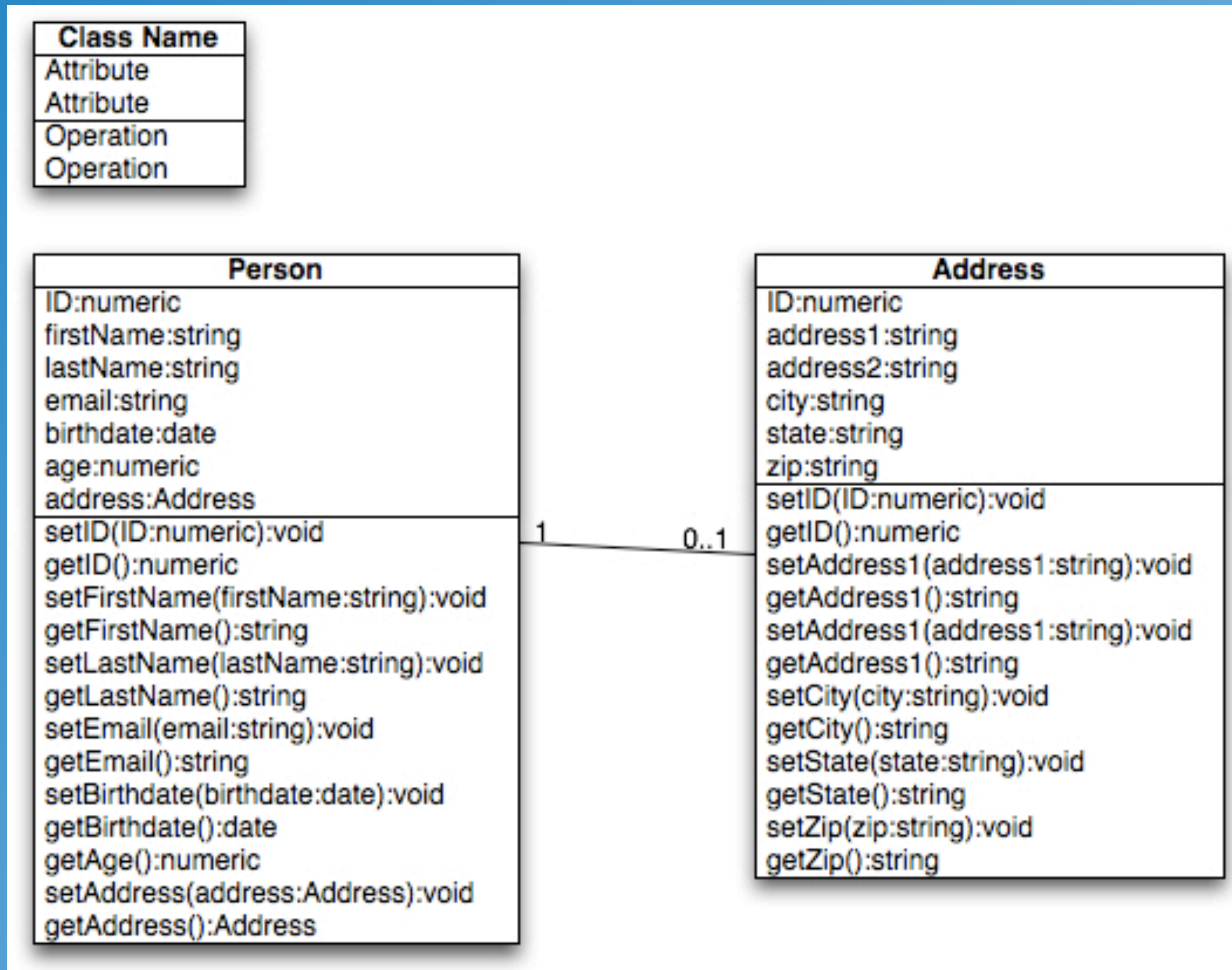
4th-6th June 2008
Edinburgh, Scotland

SCOTCH ON THE ROCKS

Thinking in Objects

- Conceptualize before you start coding!
  - easier to think about objects and their relationships when you aren't thinking about the code
- Use UML to draw objects and relationships
  - don't get any more detailed then necessary
  - don't be a slave to UML

Simple UML Example



OOP Review

- Advantages of OOP
  - applications are safer, more bug-free
  - easier to conceptualize and build large applications
  - reduces complexity
- Objects
  - software representation of real-world object
- Power of Objects
  - encapsulation and data hiding
  - data type safety and validation
  - inheritance and composition

Implementing OOP In Your Development

- Conceptualize first, code later
  - use simple UML diagrams
- Create cohesive objects
  - objects should do one thing and do it well
- Create loosely coupled objects
  - objects should depend on each other as little as possible
  - hide implementation details
  - objects should only know how to communicate with one another

Be Fearless

- Play around, break stuff, do it wrong

Questions?

- Thanks!
- Matt Woodward
  matt@mattwoodward.com
  http://www.mattwoodward.com/blog