

THE VIEW

LOTUS DEVELOPER 2006

Object-Oriented Programming in LotusScript

Bill Buchan

HADSL

© 2006 Wellesley Information Services. All rights reserved.



What We'll Cover ...

- **Introduction**

- Why use Object-Oriented (OO) methodologies?
- OO Basics in LotusScript
- Demo OO application — Directory Compare
- Extend or encapsulate?
- Large OO projects
- Pitfalls and tricks
- Wrap-up

Purpose

- This talk aims to bring a LotusScript developer forward, showing:
 - ◆ Basic Object-Oriented (OO) methodology
 - ◆ Practical OO code
 - ◆ Tips and tricks with Object Orientated Programming (OOP)
 - ◆ Large OOP project considerations

Why Object Orientated Programming?

- **Object Orientated Programming (OOP)**
 - ♦ Is the basis of modern computing environments such as Java or PHP
 - ♦ Reduces code volume
 - ♦ Aids code-reuse
 - ♦ Simplifies complex applications
 - ♦ Is easy!

Who Am I?

- Bill Buchan
- Dual PCLP in v3, v4, v5, v6, v7
- Ten+ years senior development consultancy for Enterprise customers
 - ♦ Learn from my pain!
- Five+ years code auditing
- CEO of HADSL — developing best practice tools

What We'll Cover ...

- Introduction
- **Why use Object-Oriented (OO) methodologies?**
- OO Basics in LotusScript
- Demo OO application — Directory Compare
- Extend or encapsulate?
- Large OO projects
- Pitfalls and tricks
- Wrap-up

What Is Object Orientated Programming?

- It is a way of writing large, complex systems in a more intuitive manner
- The core assumptions are:
 - ♦ Most objects will represent “actors” or “processes” in a business problem
 - ♦ You do not need to know how an object is implemented in order to use it (information hiding)



What Is Object Orientated Programming? (cont.)

- It has been in LotusScript since its inception
- It is a different way of breaking down problems
 - ♦ Compared to structured programming
 - ♦ It requires more thought during architecture
 - ♦ It requires less work during implementation (in a well-architected system)



Why Use OO Methodologies?

- **Maintainability and robustness**
 - ♦ Small code sequences are easier to maintain
 - ♦ Code hiding lends to better componentization and design
 - ♦ Classes can be tested in a stand-alone basis before integration with other components
 - ▶ It can help you test sooner (and should be considered best practice)
 - ♦ It keeps data and code for that data in a single component, thus making maintenance simpler

Why Use OO Methodologies? (cont.)

- **Maintainability and robustness (cont.)**
 - ♦ Loose coupling is better than tight coupling
 - ▶ That is, less interconnection between objects means that the interface between objects is simpler, and therefore easier to reproduce
- **Promotes re-use**
 - ♦ The holy grail of software development
 - ♦ Standard business components can be re-used
- **Look at the future**
 - ♦ Java anyone?

How to Use OO Methodologies

- Define a class

```
Class fred
  public mystring as String

  sub new()
    me.myString = "Hello World"
  end sub
end class
```

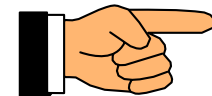
- Create one or "instances" of this class in memory

```
Dim myFred as new Fred()
print myFred.myString
```



Some OOP Generalizations

- “It’s difficult to use” — not at all
- “It’s a steep learning curve” — no
- There is no benefit
 - ♦ Untrue. For a medium to large-scale project, getting it right from the start means more code re-use, more robust applications, and less effort.
- There is one class for each item in a system
 - ♦ Probably not. Try not to be too prescriptive in any architecture.
- It takes a couple of projects before you “get it”
 - ♦ This is probably true. “The OO moment.”



Note

What We'll Cover ...

- Introduction
- Why use Object-Oriented (OO) methodologies?
- **OO Basics in LotusScript**
- Demo OO application — Directory Compare
- Extend or encapsulate?
- Large OO projects
- Pitfalls and tricks
- Wrap-up

How to Use OO Within LotusScript

- Use classes to define objects
- Use new and delete to create instances of classes
 - ◆ Think of “notesDatabase”
- They bundle together
 - ◆ Members (properties)
 - ◆ Functions and subs (methods)
- Classes can inherit design from other classes
 - ◆ In which case, they inherit all the properties and functions of the parent class
 - ◆ More on this later

OO Basics

- **Classes are defined by wrapping them with:**
 - ♦ `class <name> [as <parentClass>]`
 - ♦ `end class`
- **Classes are defined in the declarations section of a Lotuscript code sequence**
 - ♦ Remember. R5 — 64k limit. R6 — bigger. ND7 — best.
- **When a class is constructed:**
 - ♦ The sub `new()` function (the constructor) is called for all the parents of this class, and then the current class
- **When a class is deleted:**
 - ♦ The sub `delete()` (the destructor) function is called
- **You cannot extend Notes systems classes**



Note

OO Basics (cont.)

- **You can define items within a class as:**
 - ♦ Public – anything else can use this property or method
 - ♦ Private – only the class itself (or any subclasses) can use this property or method)
- **You can extend or subclass classes**
 - ♦ The class animal could be extended by class dog, which is a more precise version of the class
 - ♦ Class dog shares and re-uses all properties and methods from animal
 - ♦ Class dog may override methods within animal
 - ♦ Class dog may provide more properties and methods than class animal

Example: Animal and Dog

```
Class animal
```

```
  Private myName as String
```

Private Member



```
Sub new()
```

```
  me.myName = "Animal"
```

Constructor



```
End sub
```

```
Public function getName() as String
```

```
  getName = me.myName
```

```
end function
```

```
Public function getSpecies() as String
```

```
  getSpecies = "Animal"
```

```
End function
```

```
End class
```

Example: Animal and Dog

```
Class Dog as Animal  
  Public masterName as String  
  sub new()  
    me.myName = "Dog"  
  end sub  
End class
```

Dog inherits from class
"Animal"

Public Member

Overridden Constructor

OO Basics — Some Best Practices

- Each class contained in a different script library
- Do not expose your internal properties
 - ◊ Declare them as “private”!
- Use “Me” to resolve ambiguity
- You can use property get and set constructs
 - ◊ Functions that look like class properties
 - ◊ Advise against it
- Keep the classes small
 - ◊ Large classes mean that you should split them up
 - ▶ That you probably are not re-using things as much as you can
- Always architect before optimizing

What We'll Cover ...

- Introduction
- Why use Object-Oriented (OO) methodologies?
- OO Basics in LotusScript
- **Demo OO application — Directory Compare**
- Extend or encapsulate?
- Large OO projects
- Pitfalls and tricks
- Wrap-up

OO Basics—the Person Class

```
Class Person
```

```
    Public PersonName as NotesName
```

```
    sub new(doc as notesDocument)
```

```
        set me.PersonName = new NotesName(doc.FullName(0))
```

```
    end sub
```

```
    public function getName() as String
```

```
        if (me.PersonName is nothing) then exit function
```

```
        getName = me.PersonName.Abbreviated
```

```
    end function
```

```
End class
```

So How Do You Use This?

Use `"class:personClass"`

Library names are case sensitive on non-Wintel platforms!

```
dim doc as NotesDocument
set doc =
    vLookup.getDocumentByKey(_
        "Fred Bloggs/Acme", true)

Dim P as new person(doc)
Print "Name: " & P.getName()
```



Note 22

What We'll Cover ...

- Introduction
- Why use Object-Oriented (OO) methodologies?
- OO Basics in LotusScript
- Demo OO application — Directory Compare
- **Extend or encapsulate?**
- Large OO projects
- Pitfalls and tricks
- Wrap-up

Extend or Encapsulate?

- **Extending a class**

- ♦ The new class inherits all properties and methods from an existing class
- ♦ You can “override” original class methods
 - ▶ You cannot “overload” methods (as Java can)
- ♦ You can add new methods
- ♦ You can extend to any depth
- ♦ You can only extend from a single class
- ♦ The new class is a more precise version of the original class
 - ▶ Example: NotesRichTextItem extends NotesItem



Checklist

Extend or Encapsulate? (cont.)

- **Encapsulating a class**
 - ♦ Makes a new class more useful by containing (or encapsulating) other classes within it
 - ▶ **Example: NotesSession encapsulates an instance of NotesDatabase and calls it currentDatabase**
 - ♦ Remember: You dont have to solve the entire business problem in a single class
 - ▶ **The higher level objects will probably contain collections of lower-level objects**

Make This More Useful By Encapsulation

```
Class PersonCollection
```

```
  private people list as Person
```

```
  public sub addPerson(P as Person)
```

```
    set me.people(P.PersonName.Abbreviated) = P
```

```
  end sub
```

```
  public function findPerson(uName as String) as Person
```

```
    set findPerson = nothing
```

```
    if (isElement(me.people(uName))) then
```

```
      set findPerson = me.people(uName)
```

```
    end if
```

```
  end function
```

```
End class
```

Using PersonCollection: Loading

```
Use "class:PersonCollection"  
Dim PC as new PersonCollection  
  
Dim doc as NotesDocument  
Set doc = myView.getFirstDocument  
While Not (doc is nothing)  
    dim P1 as new Person(doc)  
    call PC.addPerson(P1)  
    set doc = myView.getNextDocument(doc)  
Wend
```

Using PersonCollection: Execution

```
Dim P as person
Set P = PC.findPerson("Joe Bloggs/Acme")
If Not (P is nothing) then
    print "found person: " & _
        P.PersonName.Abbreviated
End if
```

Improving PersonCollection

- We're placing a large code-burden on the consumer of this class
 - ◆ It may be used in a number of different places
 - ◆ It makes sense to initialise PersonCollection from a collection of Documents
 - ◆ Existing Collections include:
 - ▶ NotesDocumentCollection
 - ▶ NotesView
- Therefore, lets put another method in PersonCollection which reads from a document collection

Improving PersonCollection: LoadAllDocs

```
Class PersonCollection
```

```
...
```

```
public Function loadAllDocsInView(mV as NotesView)
```

```
    dim doc as NotesDocument
```

```
    set doc = mV.getFirstDocument
```

```
    while Not (doc is nothing)
```

```
        dim P as new Person(doc)
```

```
        call me.addPerson(P)
```

```
        set doc = mV.getNextDocument(doc)
```

```
    wend
```

```
end function
```

```
end class
```

Improving PersonCollection: Execution

```
Dim PC as new PersonCollection
```

```
Call PC.loadAllDocsInView(myView)
```

```
...
```

```
Dim P as person
```

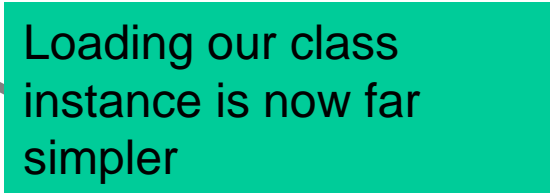
```
Set P = PC.findPerson("Joe Bloggs/Acme")
```

```
If Not (P is nothing) then
```

```
    print "found person: " &
```

```
    P.PersonName.Abbreviated
```

```
End if
```



Loading our class instance is now far simpler

Now Let's Use PersonCollection


- We'll use this example class to quickly compare two Domino directories
 - ♦ Open both directories and views in each database
 - ♦ Create a new PersonCollection from each view
 - ♦ Iterate through the first PersonCollection, and check to see that this person is in the second view
 - ♦ Vice versa
- Why does this help?
 - ♦ We only load the Person Documents once
- We can further improve this by:
 - ♦ Using NotesViewEntryCollection instead of NotesView

Loose Binding: Introduction

- “Loose Binding” means:
 - ♦ The only checking performed is at run-time
 - ♦ It only requires that the functions and properties used are supported by the calling object
- How can we use this to help us:
 - ♦ Instead of passing a NotesView or NotesDocumentCollection, pass a Variant
 - ▶ The Variant can contain an Object such as NotesView or NotesDocumentCollection
 - ▶ Since both use “GetFirstDocument” and “GetNextDocument”, the same code can be used to handle both types of input
 - ♦ We could use TypeName to validate that we can deal with this class type

Loose Binding Example

```
public Function loadAllDocsInView(mV as Variant)
  dim doc as NotesDocument
  set doc = mV.getFirstDocument
  while Not (doc is nothing)
    dim P as new Person(doc)
    me.addPerson(P)
    set doc = mV.getNextDocument(doc)
  wend
end function
```



This is the only change to the code
– and now we can deal with
NotesView as well as
NotesDocumentCollection

Demo — Directory Comparison Tool

Demo

A Directory
Comparison
using the
Person and
PersonCollection
classes



What We'll Cover ...

- Introduction
- Why use Object-Oriented (OO) methodologies?
- OO Basics in LotusScript
- Demo OO application — Directory Compare
- Extend or encapsulate?
- **Large OO projects**
- Pitfalls and tricks
- Wrap-up

Designing Large OO Projects

- **How to design large OO projects**
 - ◆ Consider various approaches, and iterate towards a good OO architecture
 - ◆ Good design and architecture is better than coding
 - ◆ Use skeleton classes to mimic behavior
 - ▶ **Allows early integration**
 - ▶ **Allows you to finalize class structure quickly**
 - ◆ Unit test each class
 - ▶ **Keep unit testing them to prevent regressions**

Design Considerations

- **It's never completely correct**
 - ♦ Beware tightly bound mutually-dependant classes!
 - ♦ Remember that specifications change over time
 - ♦ Architect for clarity and maintenance
 - ♦ Performance is an implementation consideration!

Design Considerations (cont.)

- **If you have to change class methods frequently**
 - ♦ You are exposing weaknesses in the design phase
 - ♦ You are open to regression errors
 - ♦ Once methods behavior is finalized, don't change them! Add more methods, but don't modify.
- **It might take a few OO projects before it clicks**

Tips for Large OO projects

- “Help — I don't know where this method is declared!”
 - ♦ Use a LotusScript class browser
 - ♦ Craig Schumann of Teamstudio's blog:
<http://blogs.teamstudio.com>.
- **Keep the objects simple**
 - ♦ If you have more than 15 methods, perhaps it's time to split it up?
 - ♦ If it covers more than 1,200 lines of code, perhaps it's time to split it up?



What We'll Cover ...

- Introduction
- Why use Object-Oriented (OO) methodologies?
- OO Basics in LotusScript
- Demo OO application — Directory Compare
- Extend or encapsulate?
- Large OO projects
- **Pitfalls and tricks**
- Wrap-up

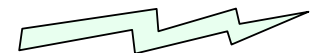
Pitfalls — Fixup Time on Load

- LotusScript is a p-code compiled, interpreted language
 - ♦ P-code and interpretation similar to Java
 - ♦ Loosely-typed — unlike Java
 - ♦ Chunks of code are “compiled” into machine-independent objects
 - ♦ These chunks of code are then dynamically linked (resolved) at run-time
- LotusScript has some “loose” data constructs
 - ♦ For example, Variant



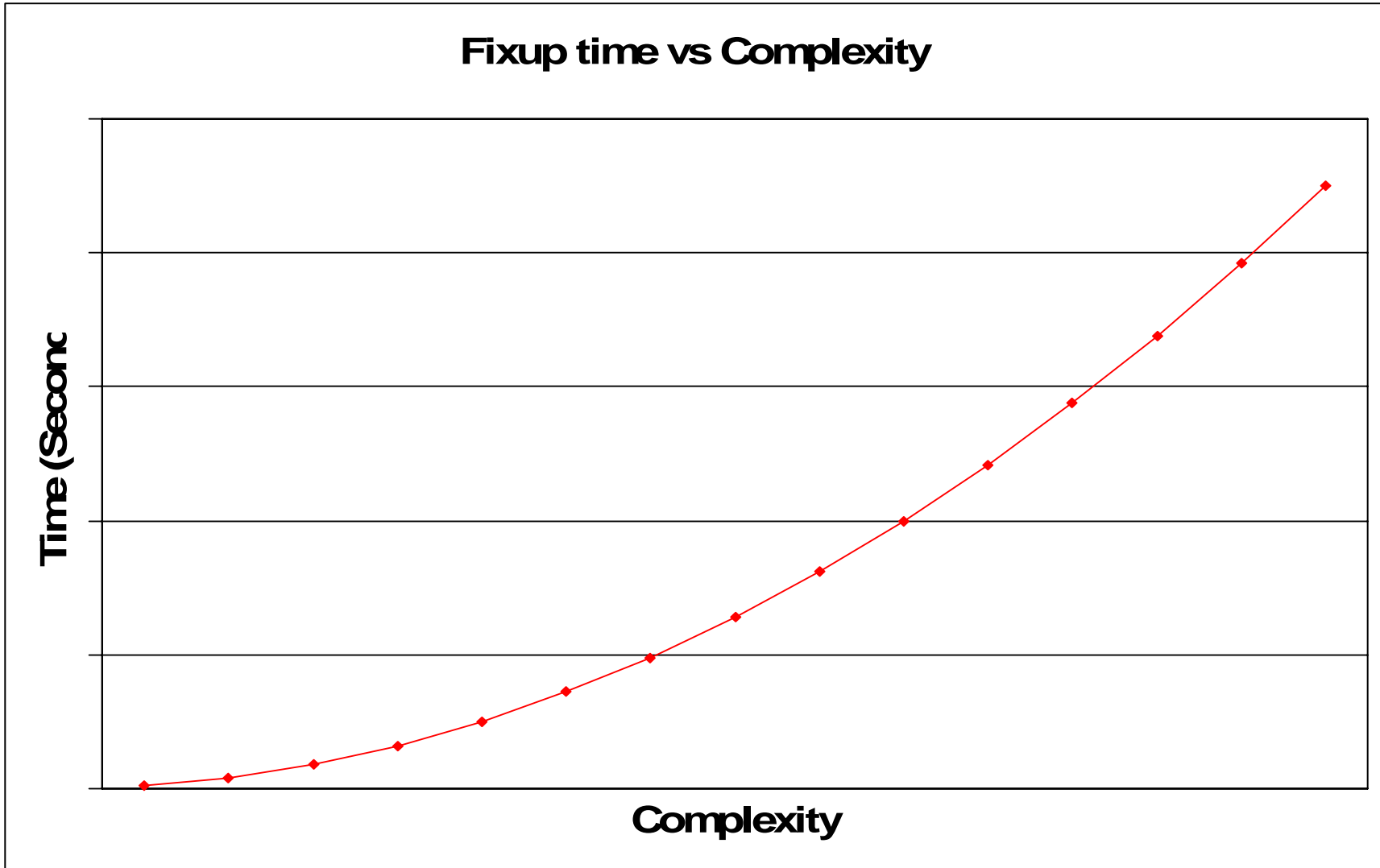
Pitfalls—Fixup Time on Load (cont.)

- This means that fixup time is significant
 - ♦ What is fixup time?
 - ▶ The time taken to load all modules and to resolve all variables
 - ▶ In traditional compiled programs, this was performed by the linker at executable build time
 - ♦ How does this affect me?
 - ▶ The more complex your applications, the longer it takes — almost geometrically — to load (this applies to ALL code, not just classes)
 - ▶ Example: 20 classes — 4 seconds load time. 120 classes — 145 seconds load time.



GOTCHA!

Fixup time?



How to Code Around It

- Fixup time doesn't have significant impact before 20+ classes and/or script libraries (in my experience)
 - ◆ My Example: 120+ classes — 145 seconds to load
- Use “loose binding” instead of “tight binding”
 - ◆ Use Variants to hold and pass classes
 - ◆ Use Dynamic Loading to instantiate classes
 - ◆ After implementation, same 120+ classes — 15 seconds
 - ◆ Downside — no more type checking!
 - ▶ Demands programming discipline and rigor!

Dynamic Loading Reference

- **Dynamic Loading is:**
 - ♦ Creating class instances at runtime based on Data, without using “use”
- **Dynamic loading is a technique outlined in Redbook**
 - ♦ “Performance Considerations for Domino Applications”
 - ♦ SG24-5602
 - ♦ Buried in Appendix B, Page 243
- **Numerous Gary Devendorf Web Services examples**



Dynamic Loading: ClassFactory

```
Class FredFactory
```

```
Public Function produce As Variant
```

```
Set produce = New Fred
```

```
End Function
```


```
End Class
```

```
Class Fred
```

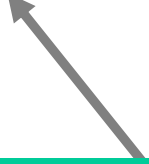
```
...
```

```
End class
```

The 'Factory' Version of this class exists only to create a new instance of the main class



There are no constructor arguments being passed to class 'fred'. This will have to be changed if you choose to add constructor arguments.



Dynamic Loading Example: CreateOtherClass

Private P as variant ` This HAS to be global!

Function createOtherClass_

(scriptName as String, className as String) _
as variant

Dim exString as String

exString = | Use "| & scriptName & |"

sub initialize

Set P = new | & className & |Factory &

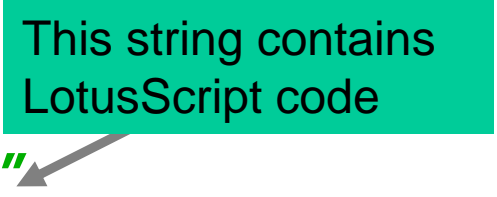
end sub |

Execute exString ` Now run the code in
exString

Set createOtherClass = P

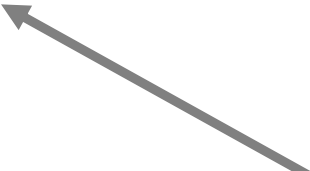
End sub

This string contains
LotusScript code



Dynamic Loading Example: Execution

```
Sub DynamicLoadingExample
  dim myClass as variant
  set myClass = createotherClass(_
    "class:fred","fred")
End sub
```

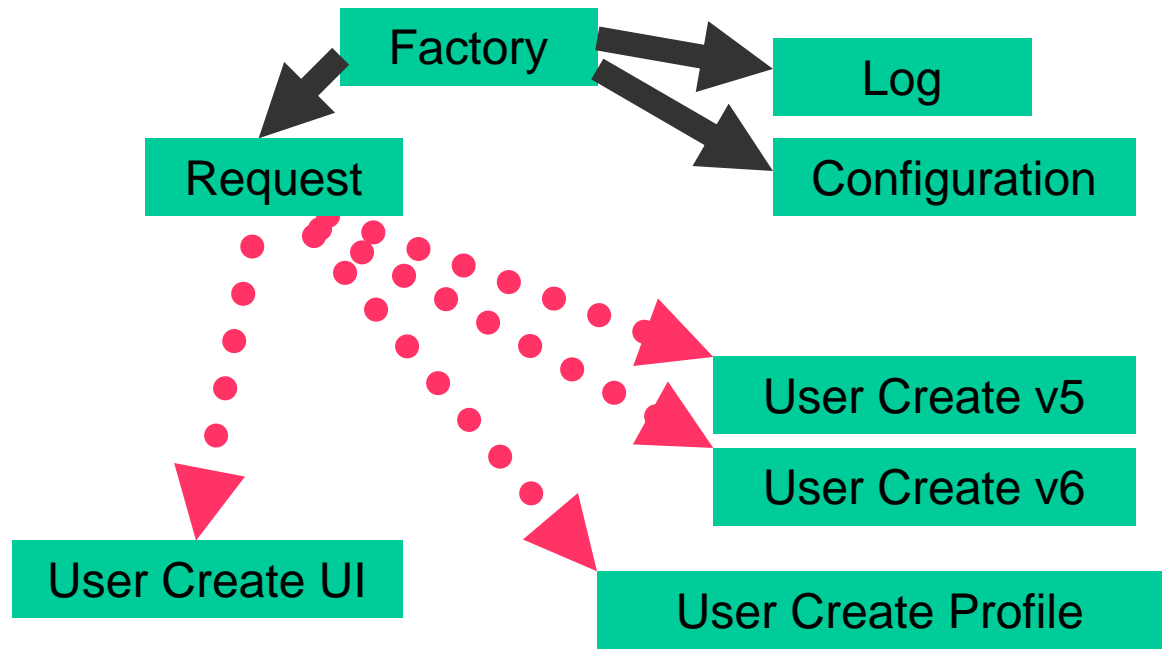


MyClass now contains an instance of Class "Fred" from script library "Class:Fred"

Dynamic Loading Architecture

- For large applications, use the “factory” pattern
- Always have one class that knows how to construct the rest without using the “use” directive
 - ♦ A factory class, for instance
- You don’t have to dynamically load all classes
 - ♦ Common classes such as “log” or “config” are always normally resident
- You can dynamically load different classes
 - ♦ Based on Notes/Domino version, platform, data
 - ♦ Using one dynamic load function means you have to use the same signature on the class constructor

Example: Version Specific Classes



Version Specific Classes

- You're loading a class you decide at run time
- You can load separate classes depending on:
 - ♦ Version, platform, capability, user permission, and your data!
- Different version classes can inherit from each other
 - ♦ So your version six class inherits from the version five class
 - ♦ Or your AIX class inherits from your core class
 - ♦ Only overrides the functions that need to be different
- Use Classes to do the work for you!

Version Specific Class: Example

```
Class createUserClass
```

```
  sub new()
```

```
  end sub
```

```
  ...
```

```
  function createUserInNAB()
```

```
    ' Version 5 specific code...
```

```
  end function
```

```
  sub createNewUser(...)
```

```
    call setupMyClass()
```

```
    call doInitialisation()
```


```
    call createUserInNAB()
```

```
    call createUserMailFile()
```

```
  end sub
```

```
end class
```

All of our version five specific code is in this particular function, for ease of overloading



Version Specific Class: Example (cont.)

```
Class createUserClassv6 as createUserClass
  sub new()
  end sub

  ...

  function createUserInNAB()
    ' Version 6 specific code...
  end function
end class
```

Our v6 Class inherits from our v5 class

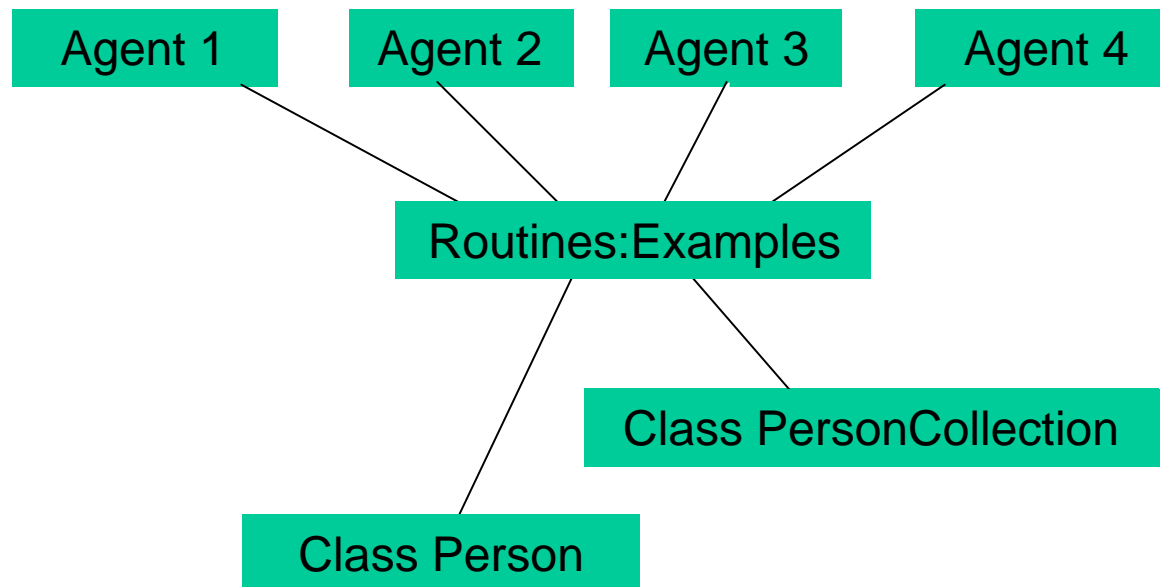
The new class ONLY contains the single overridden function, which has the version specific code contained within it

All consumers of this class see exactly the same functionality, etc.



Compilation Woes

- Expect Compilation Complexity
- For instance, consider the fairly simple class structure



Compilation Woes (cont.)

- Change a subclass and you will probably see:
 - ♦ Server Console AMGR message or client dialog
 - ▶ "11/23/2004 12:03:03 PM Agent '(Blah)' error:
Error loading USE or USELSX module: cl_event_v2
"Error %s"
- Why? The signature of the class has changed
- How to resolve?
 - ♦ Tools, Compile all LotusScript
 - ♦ Similar to Java. Most Java IDE's will rebuild all when significant change is spotted.
- Dynamic loading avoids this issue



Physical Code Hiding: Introduction

- Code hiding as a technique really refers to the ability to remove source code from the consumers of your code
 - ♦ So that your consumers (and this may be yourself) don't know how something is implemented
 - ♦ You present and work to a well-defined interface
- However, commercial programmers also require to physically hide their source code from customers



Physical Code Hiding: Execution



- The “Hide Design” method can be restrictive
- To hide code in script libraries:
 - ♦ Open the script library’s design
 - ♦ Replace item “ScriptLib” with another String
 - ▶ For example: “© MyCompany”
 - ♦ The end user cannot recompile the library — and cannot see your code
 - ♦ Warning. No way back (obviously).
 - ▶ So do this on your build copies of templates, not your development templates!



Heads Up!

What We'll Cover ...

- Introduction
- Why use Object-Oriented (OO) methodologies?
- OO Basics in LotusScript
- Demo OO application — Directory Compare
- Extend or encapsulate?
- Large OO projects
- Pitfalls and tricks
- **Wrap-up**

Resources

- Soren Peter Nielsen, et al. *Performance Considerations for Domino Applications* (IBM, March 2000), www.lotus.com/redbooks
 - ♦ Search for SG24-5602
- **Notes.net Article**
 - ♦ Bruce Perry, "Using the object-oriented features of LotusScript," (Notes.net October 2001).
 - ▶ www-128.ibm.com/developerworks/lotus/library/lso-object_oriented_LotusScript/
- **"Project: OpenDom"**
 - ♦ Managed by Alain H Romedenne
 - ▶ www.openntf.org/Projects/pmt.nsf/ProjectLookup/OpenDOM



Resources (cont.)

- **Books:**

- ♦ Steve McConnell, *Code Complete 2* (MS Press, 2004).
- ♦ Andrew Hunt and David Thomas, *The Pragmatic Programmer* (Addison Wesley Professional, 1999).
- ♦ Timothy Budd, *Understanding Object-Oriented Programming with Java* (Addison Wesley Professional, 1999).

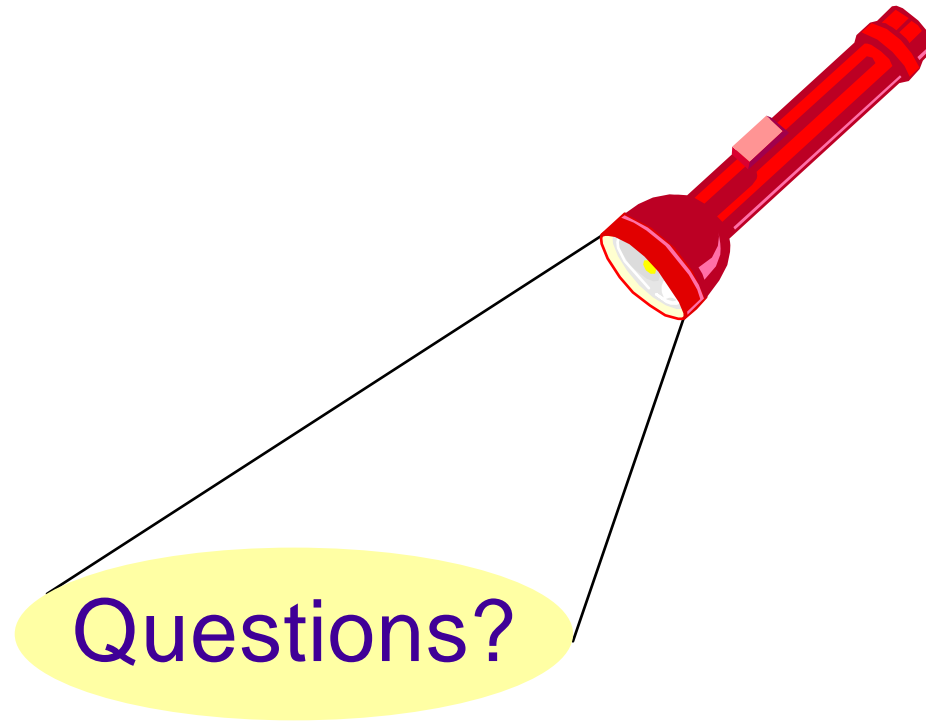
- **Example database for this presentation**



7 Key Points to Take Home

- **Object-Oriented LotusScript:**
 - ♦ Is a pain-free way of building and maintaining large Domino projects
 - ♦ Should be part of your toolset during application design
 - ♦ Can do large, serious, and complex OO projects
 - ♦ Helps get the internal data model right the first time
 - ♦ Is a stepping-stone to Java
 - ♦ Increases code-reuse, code-hiding, and maintainability
 - ♦ Decreases complexity

Your Turn!



**How to Contact Me:
Bill Buchan
Bill@hadsl.com**