


"OOP-Lates"

The Marriage of Objects and Templates

Ross A. Santos (RAS)
Executive Vice President



Introduction

- OOP provides the mechanics for defining processes and data through natural code organization. Templates provide a method to create code, over and over, reducing programming redundancy. But, neither is nirvana. However, when married, these two technologies form one of the most powerful code generation and management tools available - **"OOP-Lates"**

What's Right?

- OOP**
Reduced code, better maintenance and the ultimate in code re-use
- TEMPLATES**
Reduced coding, dictionary based generation creating the fastest database RAD solution available

What's Wrong?

- OOP
Requires coding and the knowledge of how things fit together to be productive
- TEMPLATES
Creates unnecessary code and can be too specific for generic use

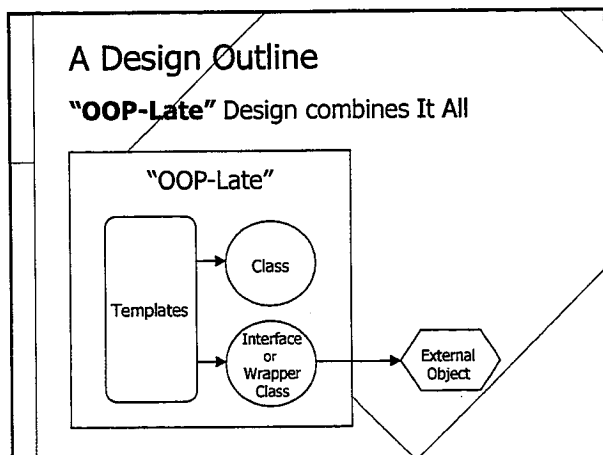
What's the Solution?

- Defining objects to replace redundant in-line code and then Wiring these objects together through "smart" and "Flexible" code generation - effectively marrying the best of both technologies to gain the biggest bang for the buck ...

"OOP-Lates"

"OOP-Lates"

- What are they made of?
 - Classes
 - Templates
 - Data
 - Processes
 - Interfaces and Wrappers



- ### Where To Start?
- Learning a new way to think about the problem space
 - Semantics
 - Building Blocks
 - Base Classes
 - Business/Application Classes
 - Interface or Wrapper Classes
 - Extension or Code Templates

- ### Semantics
- Use common terms that are accepted industry wide
 - tag , m_ , Get , Set
 - Use terms that describe the object, it's data or the action to perform
 - Address , Phone, etc.,
 - Process , Format , IsForeign, AddItem, UpdateItem, DeleteItem, etc.,
 - Whatever you do - Be consistent

Tag = ~~Type~~ Def
 M - Member Variable
 Get
 Set

Building Blocks

- OOP = Modular Development
- Common data and processes help to form the Basic Building Blocks
- Break code down into small manageable pieces that can be re-used

Base Classes

- C4 - Application Base Classes (ABC)
- A Base Class allows you to build a hierarchy among classes, gaining benefits in the descendents of the parent - for free!
- A Base Class generally provides many VIRTUAL methods so that you can "wriggle" in your own implementation at a lower point in the hierarchy - changing functionality as necessary.

Business/Application Classes

- May rely upon Base Classes
- Modularizes common tasks for re-use among applications
- May become more specific through implementation and may contain less VIRTUAL methods because of its scope of functionality.

Interface/Wrapper Classes

- Interface Classes insulate you from the gory details of the underlying object and generally move up the functionality to an application level that minimizes setup/wrap-up type of code. These type of Classes also provide a consistent and sometimes language sensitive syntax.
- Wrapper Classes are essentially Interface Classes with a more pure one-one relationship with the underlying object.

Extension/Code Templates

- Extension and Code Templates provide the "wire" or "glue" to associate controls, processes, data, etc., with a Class
- Setup/Wrap-up code should be managed by extension templates
- Method invocation should be handled by code templates
- These templates are the "visual" interface to working with the objects

What's Next?

- Understanding the "problem space" is the key factor to designing useful classes
- **Design, Design, Design ...**
- Implementation is the easy part - Its just code!

Phone Number Example (Single Line)

- Problem Space
 - Phone Numbers can have varying formats, some which will cause the CW picture statements to fail
- Issues
 - Format Localization
- Solution
 - Use a String Class and a Format Class

Address Example (Multi-Line)

- Problem
 - Users want to just type in an Address and have the program parse it apart into the appropriate fields
- Issues
 - Again, Format localization
- Solution
 - And Again, Use a String Class and a Format Class

Helpful Information

- The tagClass Definition
 - Use "typed" Classes to maximize re-use
 - Use a naming convention that illuminates ownership
 - Protect your data members
- Using a tagClass
 - You can use a tag class in two different ways; statically or through instantiation

Static, NEW and DISPOSE

- When NEWing an Object, you must also DISPOSE it.

```
myClass    tagClassX    !Static
myClass2   &tagClassX    !Instantiation

CODE
myClass2 &= NEW tagClassX
...
DISPOSE(myClass2)
```

Summary

- Consider the "Problem Space" first and foremost. Don't worry about how to do something, instead concentrate on what needs to be done.
- Use Classes to minimize code and to allow your code to be re-used from project to project.
- Use Templates to wrap Classes and to provide a visual interfaces to these Classes.

Questions...

- Fire Away...
