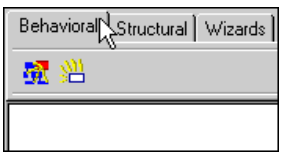
AndyPatterns

Patterns, ModelMaker & Beyond

This is a slightly older talk I gave at the time that I was into Delphi programming. It discussed using the UML tool ModelMaker in conjuntion with Delphi



ADUG Presentation
December 1999
Andy Bulka
andy@andypatterns.com

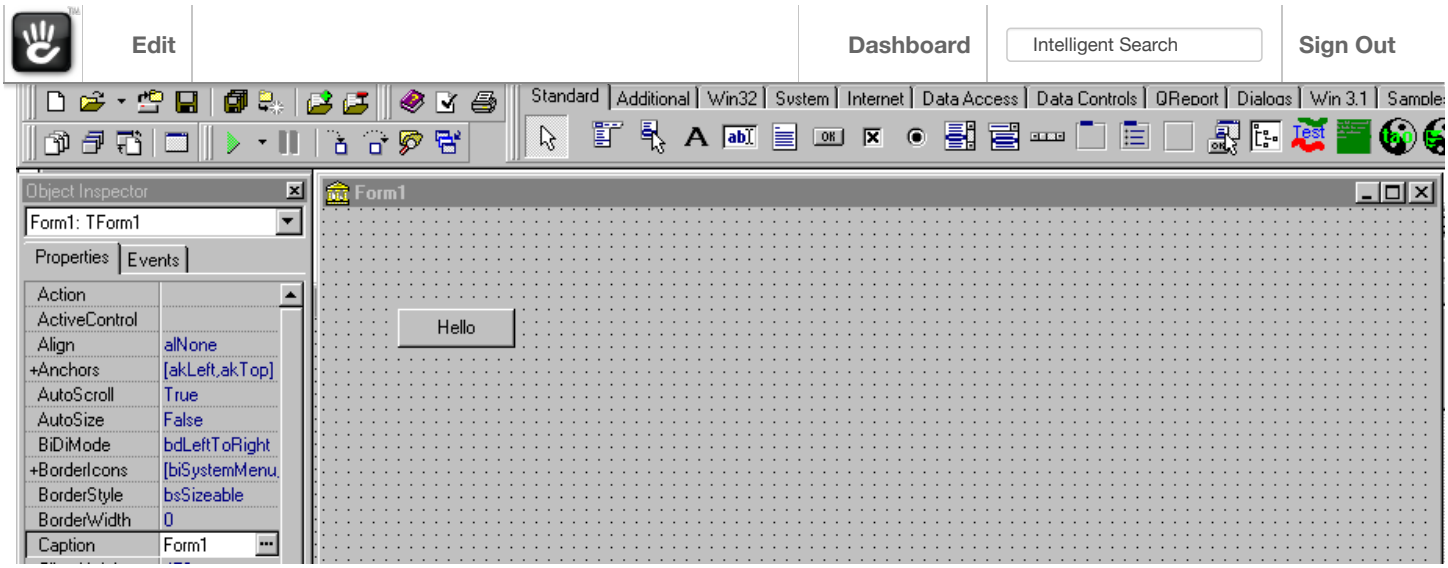
Engineers use models

- Engineers produce plans before building bridges
- Electrical engineers have a language of electrical symbols
- Architects have blueprints

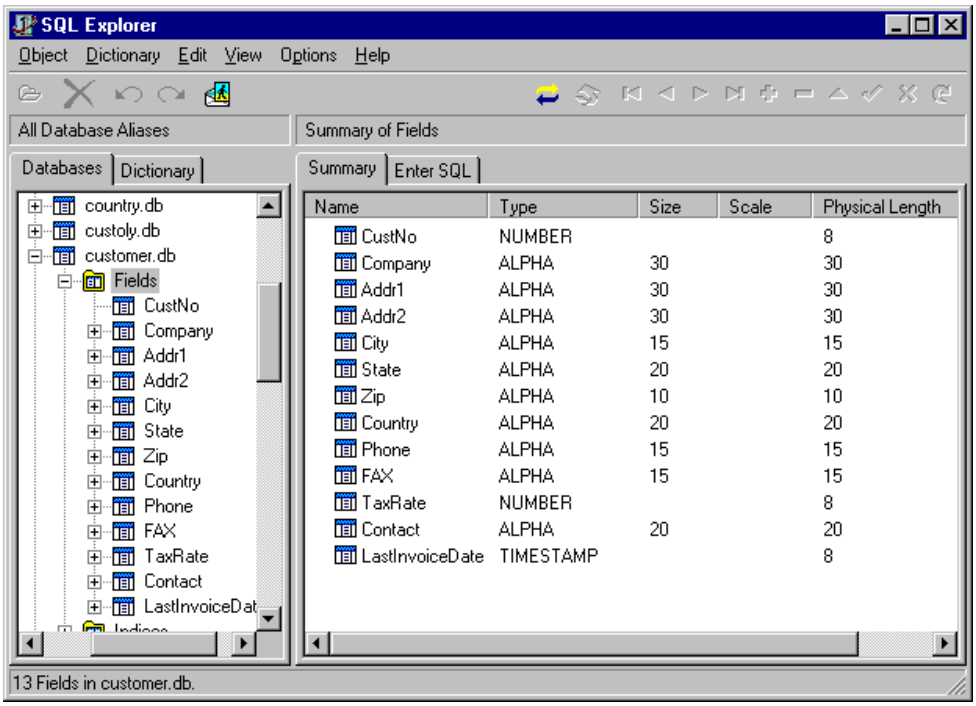
What do software engineers / programmer for a modelling tool?

Do programmers necessarily need a strict visual language like other engineers?

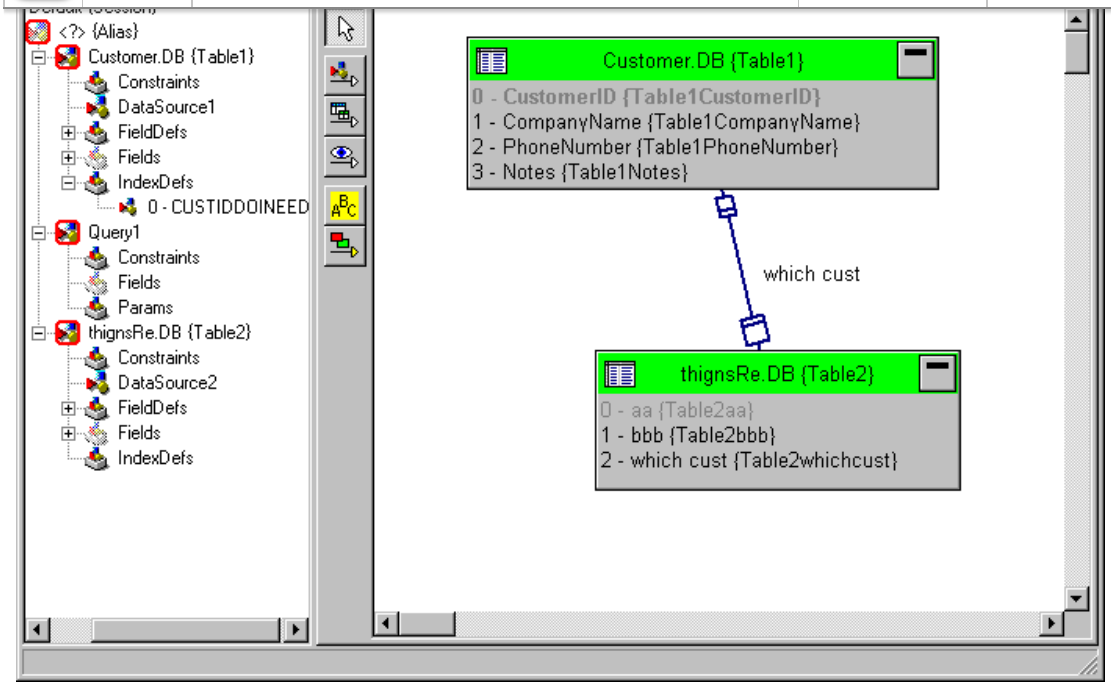
Answer: Most programmers use visual IDE (integrated development environments) to visualize and organize their work. For most programmers, this 'informal' modelling is sufficient for their needs.



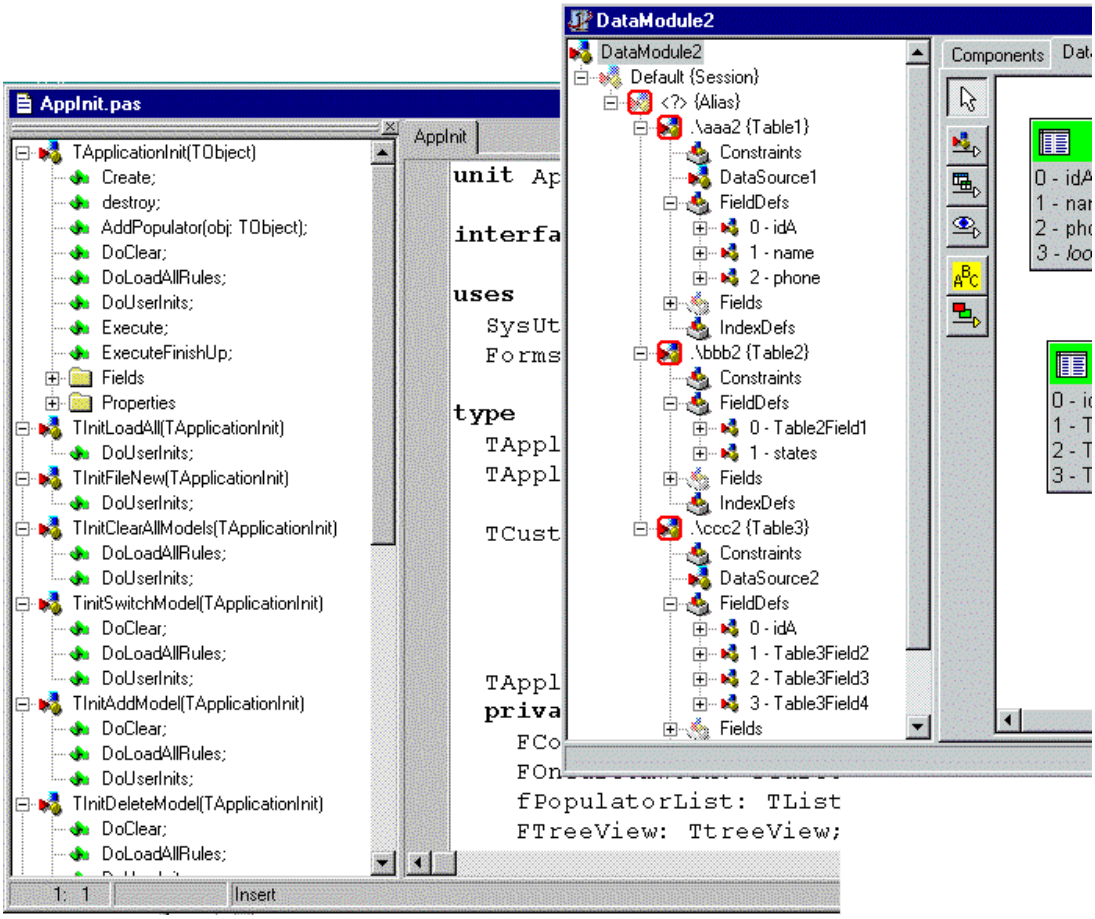
Database tables have a natural visual reprentation of a grid/table e.g.



though when you want to visualise the often complex relationships between multiple tables, you may want to use something like the Delphi 5 Data Diagram:



We also have visual modelling in the form of hierarchical representation of classes (or elements of a data module, in Delphi 5). This form of representation is extrememly useful, however, these hierarchical views do not show how objects are connected to each other.



Where does UML come in?

A modern IDE (Integrated Development Environments) like Delphi encourage a logical two-tier design where the data sits in the database and both the display & business logic sits in the presentation classes/components. Really, most programmers are filling in the blanks (e.g. event code) and setting properties - the application code hangs off the IDE / Delphi / VCL / component framework.

When someone asks you "where is your application code?" you typically answer in my GUI.

- starts to create other extra classes

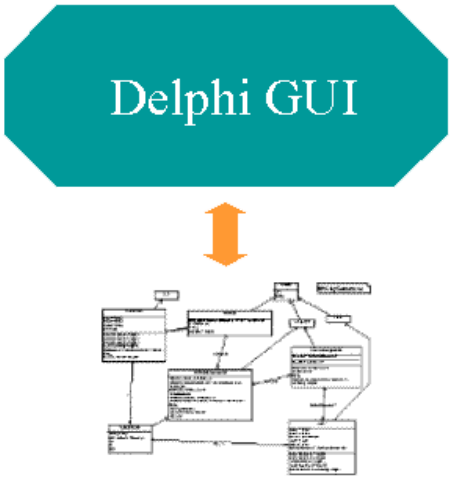
then you have no means of visualising your application code/classes. Not only are they embedded behind GUI components, but they are effectively invisible to broad visualisation and organisation.

Modelling to the rescue

Modelling software allows you to visualise your application code/classes, using UML as the visual symbol language. E.g. Objects and pointers to other objects are represented by boxes and lines..

When using a UML modeller, in addition to being able to visualize your extra application classes, you can also

- Visualize your existing Delphi components (including forms)
- Take a punt and model your data as real objects instead of DB tables - that way you can give them behaviour. They become real OO objects that model the real world domain your application is about. Your application would then be based on a proper object model.
- Keep application / business logic out of the GUI, and in a separate tier - called the model. An OO model not only can model the real world domain your application is about, but can also be the engine powerhouse of your app.

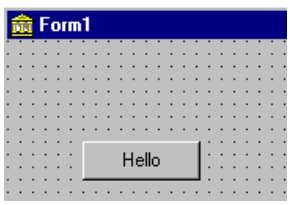


UML Model

Visualising a form in UML

A UML model can also model & incorporate GUI forms. A simple Delphi application with a form and a button on that form can be represented in the following ways:

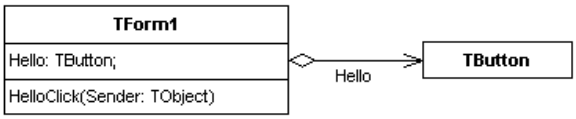
Thus



=

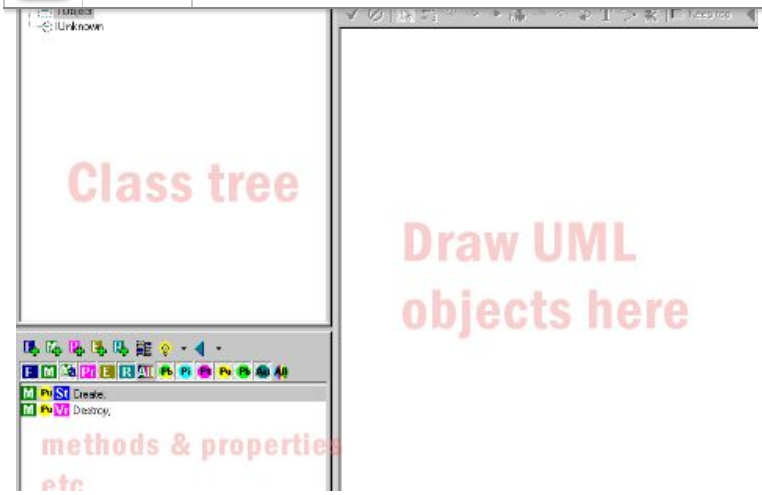
```
type
  TForm1 = class (TForm)
    Hello: TButton;
    procedure HelloClick(Sender: TObject);
  end;
```

=



Note that a form aggregates or contains a button, and this is represented by a arrow pointer. If the button's lifetime is tied to the lifetime of the form, then the arrow should have a black diamond instead of a clear one.

Modelmaker



ModelMaker benefits:

http://www.modelmaker.demon.nl/

ModelMaker represents a brand new way to develop classes and **component packages** for Borland Delphi (1/2/3/4/5). ModelMaker is a two-way class tree oriented **productivity** and UML-style CASE tool specifically designed for generating **native** Delphi code (in fact it was made using Delphi and ModelMaker). Delphi's Object Pascal language is fully supported by ModelMaker. From the start ModelMaker was designed to be a smart and highly productive tool. It has been used to create classes for both real-time / technical and database type applications. Versions 4 and 5 have full **reverse enginering** capabilities.

ModelMaker supports drawing **UML style class diagrams** and from that perspective it looks much like a traditional CASE tool. The key to ModelMaker's magic, speed and power however is the **active modelling engine** which stores and maintains all relationships between classes and their members. Renaming a class or changing its ancestor will immediately propagate to the automatically generated source code. Tasks like overriding methods, adding events, properties and access methods are reduced to selecting and clicking.

The main difference between ModelMaker and other CASE tools is that design is strictly related to and natively expressed in Delphi code. This way there is a seamless transition from design to implementation currently not found in any other CASE tool. This approach makes sure your designs remain down to earth. The main difference between ModelMaker and other Delphi code generators are it's high level overview and restructuring capabilities letting you deal with complex designs.

In ModelMaker you'll find a synergy of ideas from the cutting edge of object-oriented and component-based technology. It was inspired by the work of methodologists like Ivar Jacobson (OOSE), Ralph Johnson (design patterns) and Karl Lieberherr (adaptive software).

A unique feature, currently not found in any development environment for Delphi, is the support for **design patterns**. A number of patterns are implemented as ‘ready to use’ active agents. A ModelMaker Pattern will not only insert Delphi style code fragments to implement a specific pattern, but it also stays ‘alive’ to update this code to reflect any changes made to the design.

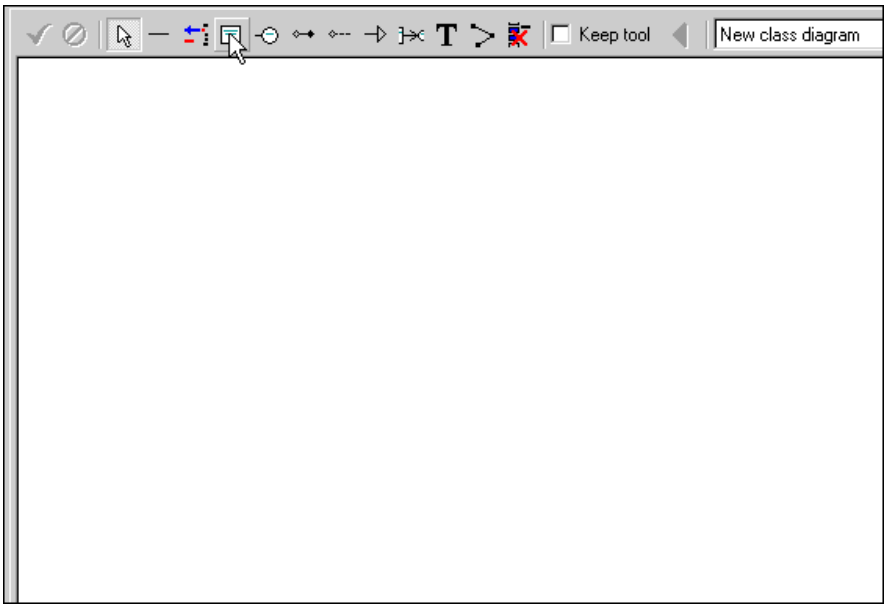
As a result, ModelMaker lets you:

- Speed up development
- Think of designing instead of typing code.
- Design without compromising and refine and experiment with designs until they feel just right.
- Create and maintain magnitudes larger models in magnitudes less time.
- Document you designs in UML style diagrams.

Simple Introduction

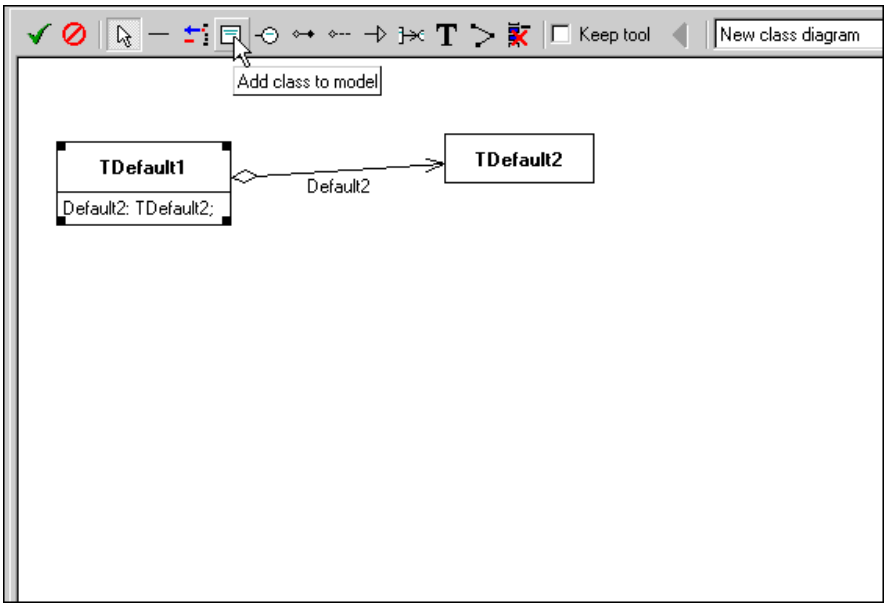
1. Creating classes
2. Adding methods and properties/fields to classes
3. Creating constructors and Destructors - MM adds the 'call inherited' code
4. Units and code generation, two way synch.

Creating a Simple Pointer property



How to create 2 new classes and have one point to the other. Also notice how a Delphi property is automatically created to represent the pointer. Of course you can add the property by clicking on the "Add property" toolbar button, but as demonstrated, it is sure sometimes conventient to be able to drag and draw a pointer - additionally, it makes it clear that the property represents a relationship between two classes, rather than the property merely holding a value or some sort.

Applying the Wrapper design pattern



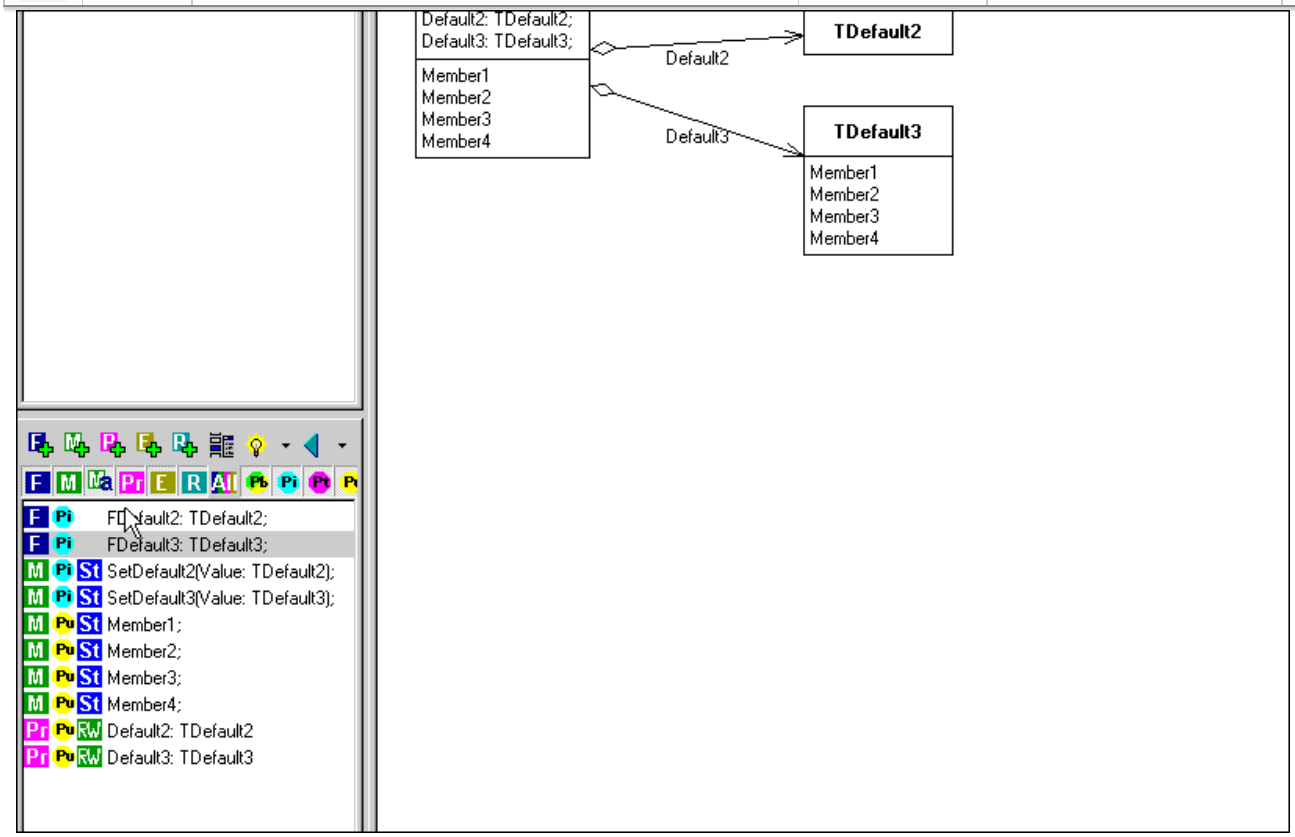
Notice we create a third class and add 4 methods to it. We want to then access these methods indirectly, via the TDefault1 class. So we duplicate the methods in the TDefault1 class, delegating / redirecting these calls to the appropriate class, where the real methods live. This technique is known as '**wrapper pattern**'. Note that the pattern will keep the method names up to date e.g. if you rename the wrapped class TDefault3 method 'Member1' into 'Member1A' then ModelMaker will automatically also rename TDefault1 class's method 'Member1' into 'Member1A'.

< template discussion here? >

Other uses of this pattern include simulating multiple inheritance (e.g. by having a class delegate out functionality to another class, you create the illusion of complex functionality coming from multiple classes).

Initialising (Create) and owning (Destroy) objects

Another use of the wrapper pattern is creating composite objects out of smaller parts. The client needs only to deal with one class, but behind the scenes there may be many classes involved. Related to 'Facade pattern'. There are issues of initialisation: Do you want the big class to create the sub classes? If so, make a **Create constructor** method in the big class



In the above demo we are creating Create and Destroy methods for the TDefault1 class, so that it both creates and destroys its 'sub-classes' when it itself is created/destroyed. After running the ModelMaker creational wizard on the TDefault1 class, the resulting Create method is

```
inherited Create;
FDefault2 := TDefault2.Create;
FDefault3 := TDefault3.Create;
```

and the resulting Destroy method is

```
FDefault2.Free;
FDefault3.Free;
inherited Destroy;
```

Note: the 'creational wizard' only generates code if it finds a create or destroy method. Also, the creational wizard only generates code for FIELDS that have been marked 'initialised' (this will generate the create code) or 'owned' (this will create the free code).

Interfaces, Patterns, Templates

After a long period of design & development, a class can end up containing many, many methods and properties. Often these methods and properties can be grouped. Wouldn't it be great if we could name and distinguish these groups? Well we already do!

Classes are an attempt to group functionality. Class A supports methods 1 & 2, Class B inherits from B and also supports methods 3 & 4.

Interfaces are another attempt to group functionality. A class can support/implement multiple interfaces.

Patterns are sets of methods and properties in a class, relating to methods and properties in one or more other classes.

Modelmaker templates are groups of methods & properties that you can save to disk and re-apply to any class. Parametrisation means you can customise the code as you apply it. You only get one shot at this. After the code template is applied, you end up with raw code in your class, which you need to maintain manually. This compares to ModelMaker's design patterns, which are more 'alive' e.g. references to a renamed method get maintained correctly within a ModelMaker pattern (e.g. Modelmaker's visitor pattern)

The Future

Working at the level of UML, 'sets/groupings of methods' and patterns, is a significantly higher level than Delphi code - which is starting to look like assembly language. One day we will be applying colored 'LEGO' blocks that represent various groups of

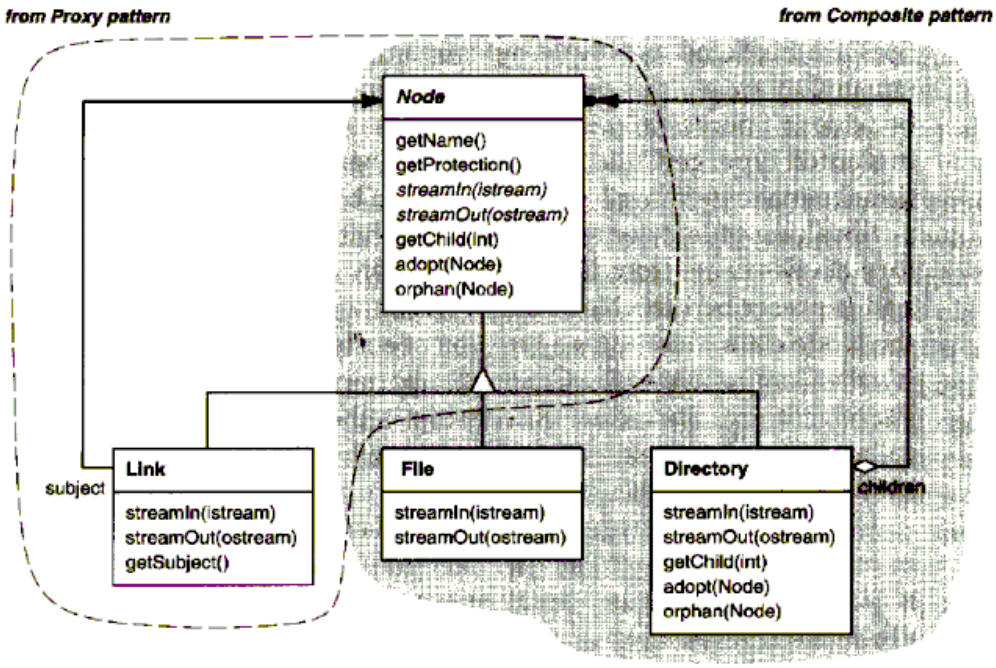
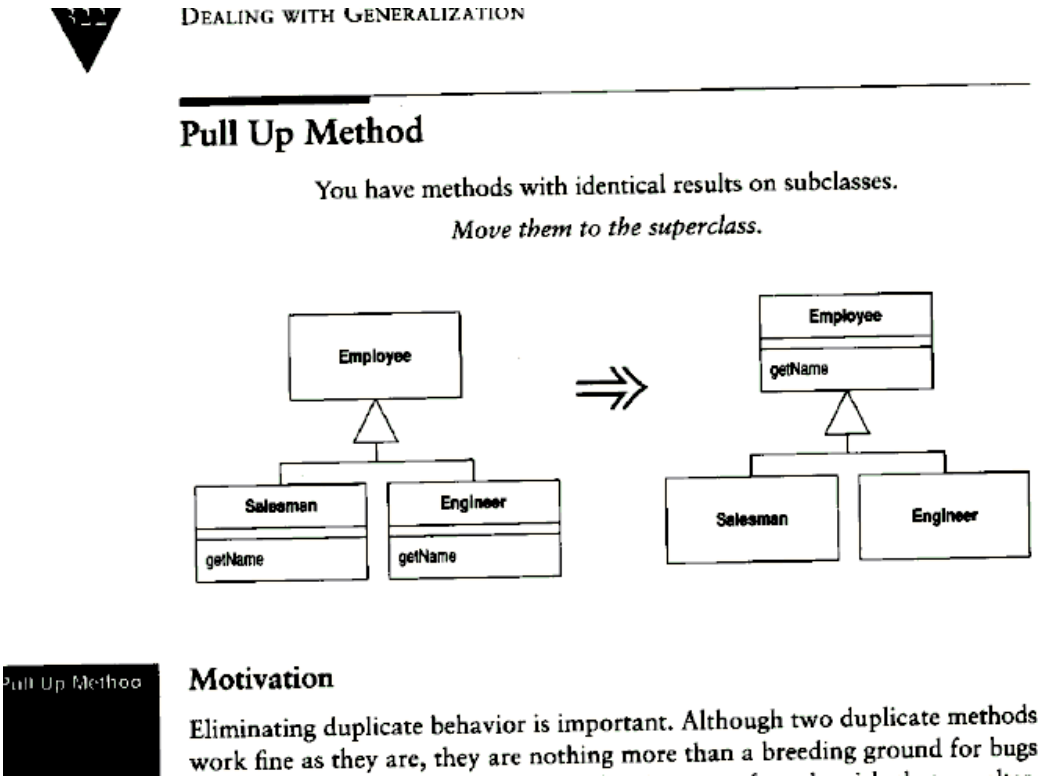


FIGURE 2.3 *Class structure embodying* COMPOSITE *and* PROXY

More ideas for what a programming environment of the future might look like: Adding a refactoring tool to ModelMaker + more patterns + templates. Sequence Diagrams auto-generated from code. Color coded groupings of patterns and methods/properties.

e.g. Here is an example of a 'refactoring' from Martin Fowler's 'Refactoring' book.:



Modelmaker can make easy work of accomplishing this refactoring using drag and drop of methods, however a built in, dedicated refactoring tool (as in Smalltalk and Java) would make the task even clearer and easier.

-Andy Bulka