Book 9

Multilanguage Object Programming

U:\Book\Book_09.doc

Multilanguage Object Programming

What to Read in This Part

Fehler! Es wurden keine Einträge für das Inhaltsverzeichnis gefunden.

1 Programming Objects In Java, VB and ABAP

Object oriented programming is a way of thinking. Their base idea is the interaction of program sniplets through messages and events. It is our goal to make you familiar with the way of thinking in distributed environments and to make it clear that object oriented programming is not programming in a different syntax where methods take the part of procedures and classes replace function libraries, but that the central philosophy of objects is the communication with messages and reacting to events. Object oriented programming reverses the traditional programming approach: instead of a big central master program to marshal program parts, there is now a community of small programs that talk to each others.

1.1 Before we start

Object oriented programming is a way of thinking

PL/1 has been one of the first object oriented languages.

I say this to everybody who claims, he had a thorough understanding of object technologies. Some are too young to know PL/1 and many others heavily deny this.

Object oriented programming is a way of thinking. Their base idea is the interaction of programs through messages and events.

PL/1 exactly fulfils this requirements, definitely better than many wannabe object programming languages like Visual Basic. OOP requires a program to communicate with parallel running processes. These processes may be instantiated by the master process or simply exist with unknown genesis. I stress this to make clear, that an OOP language is not characterised by the presence of certain language statements like “Create Object”, not 1does its absence lead to any conclusion that the language is not object oriented. ABAP IV (without objects) is the same: it is a process and transaction based language and therefore inherently object oriented.

1.2 Object Oriented Thinking

Object oriented programming is a way of thinking

To say it loud and clearly and again: Object oriented programming is a way of thinking, not a programming language. Instead of putting all the functionality into a huge lengthy program, an object based program will create simple individual program parts that expose there essential functionality to other programs (and call them *methods*.).

Traditional programs are objects, too

A traditional program can also be seen as an object, which. is by itself a compound collection of other objects, such as variables, type definitions, sub routines. Functions and subroutines in a public library are somewhat the same as methods as on object, while the library corresponds to the object class. Again: the difference is the abstract interpretation of class and library, respectively.

In programming, an object sends and receives messages

When we talk about Object programming (lets call it OOP) in a stricter sense, we basically talk about communication and messaging. If we see it that way, an object is a piece of software that can receive messages, process them and send out messages to other objects. In a true object environment an object broadcasts messages to everybody and an object that feels responsible for the requested task picks up that message, does some action and sends a message back to the original requester or broadcasts itself a result.

OOP is also the sixth programming language generation

OOP languages can also be seen as Sixth Generation Languages (1.GL: machine code; 2.GL: mnemonic languages like assembler, BASIC, FORTRAN and C; 3.GL: Typed procedural languages like PASCAL, Visual Basic and PL/1; 4:GL Query languages like SQL; 5.GL: Artificial Intelligence descriptive modeling languages like LISP, PROLOG, VRML; 6.GL: Distributed communication languages like SmallTalk, DELPHI, ADA, Java, VB.NET (maybe), C++).

Characteristics of objects implied by messaging

The following characteristics are corollaries of that universal object concept. Although them are a consequence of the base characteristic of a message based interaction they are often used as description of what objects are.

Objects are polymorph

Naturally they are, it is in the nature of messages. Polymorphism means that the same general request may be processed by different methods according to the nature of the parameters involved or the context they are in.

Example: Adding two numbers, adding two strings or two Boolean expressions are completely different algorithms although they may me denoted equally.

- 5 + 5 ➔ 10
- "Fred " + "Flintstone" ➔ "Fred Flintstone"
- true + false ➔ true (sic! Logical AND is a multiplication)

Because true objects send out messages like the following:

- Execute 5 + 5 where both operands are integer numbers

The methods that care for strings or Boolean operations will not feel addressed. For the programmer the operations will appear polymorph (will look the same although the context is different).

Inheritance

Objects can inherit methods and characteristics from another object, known as a parent object. Thinking in true abstract objects, this is more than natural: an object that descends from a parent object inherits the parent's features unless it re-implements them. This is the nature of inheritance in OOP. In technical terms, inheritance can be simple realized as a redirection. If a message is received and there does not exist a proper method, it will be redirected to the proper method of the parent object.

Instantiation

Objects have a limited lifetime. They are created and destroyed during the execution of a master program or when a special event occurs. The created object is called an instance while the template which is used as a blueprint for the object instance is called a class. Every time an object is created it typically executes a constructor (in Visual Basic: Sub Class_Initialize) and before it is destroyed a dedicated exit routine, called a destructor in Visual Basic: Sub Class_Terminate) is called.

1.3 Object Classes and Instances

Classes are blueprints for objects. Unlike traditional programs, an object instance does load the program code only once with the class while given each instance a private data memory area.

Classes are blueprints for an object creation and multiple objects can be created simultaneously from the same class blueprint

A class is a definition that describes how an object should look like and what methods it can execute. The "trick" with classes is, that an arbitrary number of objects can be created from a class template simultaneously. In classical programming terms a class would be somewhat like a program stored on a disk. Every program has its own memory space where it stores the content of its variables. An instance of a class is called an object. That would correspond to having the program with its own proper memory space loaded. Now you could load the program with a new and separate memory space loaded again and executed. That would be the second instance of a class.

An object instance is a closed memory area created from the object class template

A class is not yet an executable object but it is used to create an arbitrary number of object instances on demand. Such object instances acquire an own private memory block where the execute and store there main memory data. This memory block is completely shielded from other programs which can only communicate with the object through the predefined interfaces and methods. This is also widely known as data encapsulation.

Classical programs load the code for every instance that is called

The difference between objects and conventional structured programs is the concept of multithreading. In a conventional environment you can execute a

program by loading it into the memory, reserving a private workspace for it and execute it. You can call the program again and thus load another copy of the program into the memory. This is fine because every instance of the program gets its own memory space and data area. The bad news is, that the program is loaded over and over again into the memory, thus unnecessarily consuming precious resources.

Object load the class only once for any number of instances

Objects instances actually replicate the data block every time an instance is created while the executable code remains in the object class block and is loaded only once into the memory.

Object load the class only once for any number of instances

It actually sounds profane that loading the program code only once into the machine while giving each object instance an own memory area to work with. It truly is like that. However, writing a compiler and operating system that actually supports this type of memory management is far from trivial. In the case of Visual Basic and Windows it must be said, that it does not truly implement all these kind of memory management and threading mechanism as it is done by most UNIX and LINUX implementations. For an application developer this is not of much importance but threading concepts will be an issue for any performance and sizing analyst.

1.4 Building Classes with Visual Basic

Visual Basic stores compiled classes that belong together in libraries, the dll-files. With the CreateObject command the class template is instantiated into a living object.

Classes are blueprints for an object creation and multiple objects can be created simultaneously from the same class blueprint

A class is a definition that describes how an object should look like and what methods it can execute. The "trick" with classes is, that an arbitrary number of objects can be created from a class template simultaneously. In classical programming terms a class would be somewhat like a program stored on a disk. Every program has its own memory space where it stores the content of its variables. An instance of a class is called an object. That would correspond to having the program with its own proper memory space loaded. Now you could load the program with a new and separate memory space loaded again and executed. That would be the second instance of a class.

Steps to create a class in Visual Basic

After choosing "New project" in Visual-Basic select "ActiveX-dll" and

- enter a name for the library in the menu-item "project", "properties", "project name". This name will also be a default for the name of the VB project file (vbp) and the dll-filename. You could use three different names, but I don't know what this would be good for.
- enter a name for the class in the properties-windows. In the project-explorer you can add additional classes. The source-Code for each class are stored in a files with the extension .CLS
- select the property "MTS-TransactionMode" to anything else but "not an MTS object". "uses transactions" is a good choice.
- In the "general", "declaration"-section of the code-window you can create properties for the class by simply entering: PUBLIC XXX AS TTT (XXX is any name for the property and TTT is the data type i.e. integer.
- the prefix PUBLIC before the name of a sub-program is used to define a method.

Note that only complied dll can go into the MTS. You build the dll in the file-menu.

Listing 1: Sample Code for a class

```
Public counter as Integer
Public Sub inc()
  counter = counter + 1
End Sub
Public Sub clear()
  counter = 0
End Sub
```

⇨

1.5 Building Classes with ABAP

ABAP supports the concept of Classes since release 4 and since 4.6 they are integral part of the ABAP language and repository.

Steps to create a class in Visual Basic

Creating a local class that can be used within your currently running program is supported by means of After choosing "New project" in Visual-Basic select "ActiveX-dll" and

- enter a name for the library in the menu-item "project", "properties", "project name". This name will also be a default for the name of the VB project file (vbp) and the dll-filename. You could use three different names, but I don't know what this would be good for.
- enter a name for the class in the properties-windows. In the project-explorer you can add additional classes. The source-Code for each class are stored in a files with the extension .CLS
- select the property "MTS-TransactionMode" to anything else but "not an MTS object". "uses transactions" is a good choice.
- In the "general", "declaration"-section of the code-window you can create properties for the class by simply entering: PUBLIC XXX AS TTT (XXX is any name for the property and TTT is the data type i.e. integer.
- the prefix PUBLIC before the name of a sub-program is used to define a method.

Note that only complied dll can go into the MTS. You build the dll in the file-menu.

Listing 2: Sample Code for a class

```
Public counter as Integer
Public Sub inc()
  counter = counter + 1
End Sub
Public Sub clear()
  counter = 0
End Sub
```

⇨

1.6 Building Classes With Java

1.7 Sharing Resources and Proxy Applications

A proxy is an application that installs itself between an application and a server. For both ends the proxy will appear as the other peer.

A Proxy plays an important roles in sharing critical and expensive connections

The most obvious use of a proxy application to share a resource, like connecting through a modem to a server. In an apartment threaded model this would lead to the undesirable situation that every new session would try to dial up an individual connection to the server and result in extra phone costs. A proxy application can work as a handler to share the single phone line, once established. The proxy will look, to the client, like the server application itself and take requests from the client. The requests will then be serialized by the proxy and forwarded to the true server as soon as it is available.

Proxy marshaling

The ability to accept requests from multiple clients and redirect them to a single

connection is called *Proxy Marshaling*. Proxy marshaling commands the COM application to not create a new object instance whenever a new thread is opened but instead to serialize them and process them one after the other.

A Proxy may help to limit the number of simultaneous connections to R/3

R/3 allows a virtually unlimited number of simultaneous RFC connections. However, there are situations when you prefer to use a single connection to R/3 and re-route all requests through that single connection. This may already be desirable when your R/3 server does limit the number of simultaneous RFC connections by a single user.

A Proxy can shield the internal login data from ASP

Another even more important use of a proxy component is the ability to shield the R/3 login data from the ASP tier. For that purpose, the ASP will always contact the proxy component as a trusted application, i.e. it will not require any kind of login. The proxy will automatically perform the appropriate logon to R/3 and get the necessary logon credentials from a hidden source. The logon can be set up in a way that it remains absolutely transparent to the ASP, i.e. the ASP then can see the R/3 as a permanent and local connection.

1.8 Constructors

A constructor method is a method, which is automatically executed when a class instance is instantiated. Thus it is the last action performed during the lifecycle of a class instance.

Constructors in Java

A Java Constructor looks like a method definition, with the two defining characteristics:

A constructor has the same name as its class
It has no result type

Constructor in Java

```
public class Test {
```

This is the constructor for the class Test:

```
  public Test() {
  }
```

Here the class is created and the constructor is executed during instantiation.

```
  public static void main(String[] args) {
    Test test1 = new Test();
  }
}
```

Constructors in ABAP Objects

Constructors in VB

A constructor in Visual Basic is a special Method with the name Class_Initialize.

Constructor in Visual Basic

```
VERSION 1.0 CLASS
Attribute VB_Name = "TEST"
```

This is the constructor for the class Test:

```
Private Sub Class_Initialize()

End Sub
```

This is the destructor for the class Test:

```
Private Sub Class_Terminate()

End Sub
```

When the class is created the constructor is called automatically.

```
Set newInst = New Test()
```

Another way to create an instance in VB:

```
Set newInst = CreateObject("Test")
```

1.9 Destructors

Destructors are special methods, which are called when the instance of a class is destroyed. Thus it is the last action performed during the lifecycle of a class instance.

Destructors in Java

There are no special destructor methods in Java.

Destructors in ABAP Objects

Destructors in VB

A destructor in Visual Basic is a special Method with the name Class_Terminate.

Destructor in Visual Basic

```
VERSION 1.0 CLASS
```

This is the constructor for the class Test:

```
Private Sub Class_Initialize()

End Sub
```

This is the destructor for the class Test:

```
Private Sub Class_Terminate()

End Sub
```

1.10 Property Declarations

A property of a class is an enhancement of a variable. In addition to simply assigning or reading the value of a variable, a property allows the execution of a full sequence of actions, both when setting and reading the value of the property.

Properties in Java

Properties in Java are declared as a method with a special naming convention:

The method's name is:

- the property's name with the first character capitalised
- prefixed with "get" for a get property
- prefixed with "set" for a set property

Get Properties in Java

Set Properties in Java

Properties in VB

Get Properties in VB

Set Properties in VB

Properties in ABAP Objects

Get Properties in ABAP

Set Properties in ABAP

1.11 Methods Declarations

Methods in Java

Method in Java

Methods in VB

Method in VB

Methods in ABAP Objects

Method in ABAP

1.12 Strings Declarations

Strings in Java

String in Java

Strings in VB

String in VB

Strings in ABAP Objects

String in ABAP

1.13 Array Declarations

Arrays in Java

Array in Java

Arrays in VB

Array in VB

Arrays in ABAP Objects

Array in ABAP

245

2 Exploring SAP R/3 BAPIs With R/3

3 Examples of Calling BAPIs From Visual Basic

4 Examples of Calling BAPIs From Java

250