ABAP IV.
ORIENTACIÓN A OBJETOS, UNA VISIÓN GLOBAL

INDICE

PRÓLOGO

Este libro contiene una explicación sencilla para comprender la orientación a objetos apoyándonos en el lenguaje ABAP IV de programación de SAP.

Nuestra idea es que, mediante una descripción amena y de fácil lectura, tener una visión global de la programación orientada a objetos.

Para ello utilizamos una serie de sencillos ejemplos que nos guíen en nuestro recorrido de los distintos conceptos de la Programación Orientada a Objetos en ABAP.

Los ejemplos de código incluidos han sido probados en un sistema SAP/R3 IDES ECC 6.0 y están pensados para poder copiarlos y pegarlos directamente y que se activen sin errores, pudiendo verificar su ejecución y poder depurar el código y que nos sirvan para comprender en su totalidad cada ejemplo.

Para un mayor aprovechamiento de este libro se recomienda tener unas nociones básicas del ABAP WorkBench y de la programación de Reports, Grupos y Módulos función, Selection Screens y Module Pool, aunque no es imprescindible.

AGRADECIMIENTOS

Quiero dejar aquí mi agradecimiento a Jorge, Chema y Rogelio por su continuo apoyo desinteresado y buenos consejos en una etapa de reorientación y crecimiento tanto profesional como personal. Muchas gracias, siempre podréis contar conmigo.

1 ABAP ORIENTADO A OBJETOS

1.1 INTRODUCCIÓN

La programación Orientada a objetos (POO) es un estilo de programación (un paradigma de programación) más cercana a como expresaríamos las cosas en la vida real que otros tipos de programación. Usa “objetos” en sus interacciones, para diseñar aplicaciones y programas informáticos.

Con la POO tenemos que aprender a pensar las cosas de una manera distinta, para escribir nuestros programas en términos de “objetos”, “propiedades” (o “atributos”), “métodos” y otros elementos que veremos a continuación para aclarar conceptos de este tipo de programación.

Uno de los objetivos de la POO es la reutilización, ya que no tenemos por qué reescribir código si éste ya existe, pudiendo usarlo tal y como ya existe o ampliarlo y adecuarlo a nuestras necesidades.

El concepto de POO es un concepto (técnica) de programación, a las personas que no sepan programación les será más fácil aprenderlo y aplicarlo, las personas que conozcan otras metodologías de programación deberán olvidar las técnicas y métodos aprendidos y aprender a pensar “en objetos” y asimilar los conceptos que detallaremos a continuación.

Todas las propiedades y métodos comunes a los objetos se encapsulan o agrupan en clases. Una clase es una plantilla, un prototipo para crear objetos; en general, se dice que cada **objeto** es una **instancia** de una clase.

1.2 CÓMO SE PIENSA Y TRABAJA EN POO

Pensar en términos de objetos es muy parecido a cómo lo haríamos en la vida real. Por ejemplo vamos a pensar en un coche para tratar de "modelizarlo" (escribir el código asociado) en un esquema de POO. Diríamos que el coche es el elemento principal que tiene una serie de características, o atributos, como podrían ser el color, el modelo o la marca. Además tiene una serie de funcionalidades asociadas, o acciones que se pueden realizar con él, como pueden ser ponerse en marcha, parar o aparcar.

Pues en un esquema POO el coche sería el objeto, los atributos serían las características como el color o el modelo y los métodos serían las funcionalidades asociadas como ponerse en marcha o parar.

Por poner otro ejemplo vamos a ver cómo modelizaríamos una fracción, es decir, esa estructura matemática que tiene un numerador y un denominador que divide al numerador, por ejemplo 3/2.

La fracción será el objeto y tendrá dos atributos, el numerador y el denominador. Luego podría tener varios métodos como simplificarse, sumarse con otra fracción o número, restarse con otra fracción, etc.

Estos objetos se podrán utilizar en los programas, por ejemplo en un programa de matemáticas harás uso de objetos fracción y en un programa que gestione un taller de coches utilizarás objetos coche. Los programas Orientados a objetos utilizan muchos objetos para realizar las acciones que se desean realizar y ellos mismos también son objetos.

En la POO se trabaja con objetos, pero lo que nosotros escribimos (nuestro código) son CLASES, por ejemplo, nuestro primer programa constará de una porción de código que especificará el nombre de la clase, p. e. "coche", especificaremos que tiene los atributos color y marca y los métodos arrancar, acelerar, frenar y apagar. Eso será nuestra especificación de la clase "coche". En otra porción de código

escribiremos nuestro programa, en él crearemos un objeto “coche” usando la clase escrita previamente y le asignaremos valores a sus atributos, una vez hecho esto nuestro coche existirá y podremos realizar con él las acciones especificadas como métodos de la clase, en nuestro caso: arrancar, acelerar, frenar y apagar.

Siempre trabajaremos con los objetos, no con clases. Las clases nos sirven para crear objetos, que será con los que realmente trabajemos. Podríamos decir que tendremos (al menos) dos partes diferenciadas, una primera donde definimos nuestra clase (atributos y métodos) y otra donde creamos nuestros objetos y los usamos, que será nuestro programa.

1.3 CONCEPTOS FUNDAMENTALES

1.3.1 Clases

Una clase es una entidad teórica que describe las propiedades y comportamiento de un objeto. Es la plantilla que usaremos para crear objetos, un objeto es una *instancia* en tiempo de ejecución de una clase. Se pueden crear cuantos objetos se necesiten basados en una clase.

Cada instancia de una clase (objeto) tiene su propia identidad y su propio conjunto de valores para sus atributos. Dentro de un programa un objeto es identificado por su referencia, la cual le proporciona un nombre que define inequívocamente al objeto y permite acceder a sus métodos y atributos.

1.3.2 Objetos

Son las instancias de una clase, los datos constituyen los *atributos* del objeto. Los servicios que proporciona el objeto se conocen como *métodos* y se asemejan en su funcionamiento a las funciones de la programación procedural.

1.3.3 Atributos

Son las características, propiedades de la clase.

1.3.4 Métodos

Desde el punto de vista del comportamiento, es lo que el objeto puede hacer. Un método puede producir un cambio en las propiedades del objeto. Los eventos son métodos especiales, serán explicados más adelante.

1.3.5 Encapsulación

Los objetos restringen la visibilidad de sus recursos (atributos y métodos) al resto de usuarios. Cada objeto posee una *interface* que determina la manera de interactuar con él. La implementación del objeto (su interior) es encapsulada, lo que quiere decir que desde fuera el objeto es invisible, simplemente se usa.

Normalmente los métodos operan con los datos *privados* del objeto, esto es, con datos que son sólo *visibles* para los métodos del objeto. Así se garantiza la consistencia interna del objeto.

1.3.6 Herencia

Las clases no están aisladas, sino que se relacionan entre sí, formando una jerarquía de clasificación, como un árbol genealógico. Se pueden utilizar clases existentes para originar nuevas clases. Las nuevas clases originadas heredan los datos y los métodos de la *superclase (o clase padre*). De cualquier manera, se pueden sobrescribir los métodos existentes, incluso añadir métodos y atributos nuevos.

1.3.7 Polimorfismo

El polimorfismo quiere decir que métodos que se llaman exactamente igual pueden comportarse de manera distinta en clases diferentes. La orientación a objetos tiene unas estructuras llamadas interfaces que permiten acceder a métodos con el mismo nombre en diferentes clases.

Dentro de distintas clases puede existir un método con el mismo nombre que realice acciones distintas. Por ejemplo, el método pintar de la clase cuadrado, nos mostrará un cuadrado por pantalla y el método pintar de la clase circulo nos mostrará un circulo, ambas clases, podrían ser hijas de la clase figura.

1.3.8 Recolector de Basura

La recolección de basura o “garbage collector” es una técnica por la cual el entorno de objetos se encarga de destruir automáticamente, y por tanto desvincular la memoria asociada, los objetos que hayan quedado sin ninguna referencia a ellos. Esto significa que el programador no debe preocuparse por la asignación o liberación de memoria, ya que el entorno la asignará al crear un nuevo objeto y la liberará cuando nadie lo esté usando.

2 OBJETOS Y CLASES EN ABAP

ABAP (Advanced Business Application Programming) es un lenguaje de programación para desarrollar aplicaciones en el sistema SAP R/3. La última versión, **ABAP OO**, es programación Orientada a Objetos. SAP ejecuta aplicaciones programadas tanto en **ABAP** como en **ABAP OO.** Este es un nuevo concepto introducido desde la versión 4.0, tiene dos significados, por un lado se refiere al entorno de ejecución ABAP y por otro a la extensión orientada a objetos del lenguaje ABAP

2.1 EXTENSIÓN ORIENTADA A OBJETOS DE ABAP

ABAP Objects es en sí mismo un conjunto de sentencias orientadas a objetos que han sido introducidas dentro del lenguaje ABAP. Esta extensión se cimenta en el lenguaje ya existente, siendo compatible con él. Se pueden usar objetos en programas existentes, de la misma manera que se pueden usar sentencias ABAP convencionales en programas ABAP orientados a objetos.

El resto del lenguaje ABAP está creado desde un principio orientado a una programación estructurada, en la cual los datos se almacenan de manera estructurada en tablas en la base de datos y los programas mediante funciones acceden a estos datos y trabajan con ellos.

2.2 GRUPOS / MÓDULOS FUNCIÓN y OBJETOS

Lo más parecido a los objetos que tenía ABAP eran los módulos y los grupos de funciones. Vamos a realizar un ejemplo usando un grupo de funciones y luego vamos a realizar su equivalente usando POO.

Supongamos que tenemos un grupo de funciones para procesar pedidos. Los atributos de un pedido son los datos globales del grupo de funciones, mientras que los módulos de funciones son las acciones que manipulan los datos, o sea los métodos. Esto quiere decir que los datos reales del pedido están encapsulados en el grupo de funciones y no se puede acceder directamente a ellos, sólo mediante los módulos de funciones. De esta manera se garantiza la consistencia de los datos.

La instancia de un grupo de funciones en el área de memoria de la sesión interna representa prácticamente el concepto de objeto. Cuando se llama al módulo de funciones, el programa que llama usa la instancia del grupo de funciones basada en su descripción en la biblioteca de funciones. El programa no puede acceder a los datos en el grupo de funciones directamente pero si a través del módulo de funciones. El módulo de funciones y sus parámetros son la interface ente el grupo de funciones y el usuario.

La principal diferencia entre la verdadera orientación a objetos y los grupos de funciones es que mientras que un programa puede trabajar simultáneamente con varios grupos de funciones, no puede hacerlo con varias instancias de un mismo grupo. Si un programa quiere procesar varios pedidos a la vez tendría que adaptar el grupo de funciones para incluir una administración de instancias, usando por ejemplo, números que diferencien las instancias. En la práctica, esto es muy complicado de realizar. Por esto, los datos son almacenados en el programa y los módulos de funciones son llamados para trabajar con ellos (programación estructurada). Un problema es por ejemplo que todos los usuarios de un módulo de funciones deben usar las mismas estructuras de datos así como el propio grupo de funciones.

Esto se consigue con la orientación a objetos. ABAP Objects permite definir datos y funciones en clases en lugar de en grupos de funciones. Usando clases, un programa ABAP puede trabajar con cualquier número de instancias (objetos) basados en la misma plantilla. En lugar de cargar en la memoria una única instancia de un grupo de un grupo de funciones

implícitamente cuando se llama al módulo de funciones, el programa ABAP ahora puede generar las instancias de la clase explícitamente usando la nueva sentencia ABAP: **CREATE OBJECT**. Cada instancia representa a un único objeto, y se puede acceder a cada una mediante su referencia. La referencia del objeto es lo que permite a un programa ABAP acceder a la interface de la instancia.

Una referencia es básicamente un puntero a una dirección de memoria. Cuando creamos un objeto lo que estamos haciendo es reservar un espacio en memoria para grabar los datos asociados a dicho objeto, sus atributos y métodos, cada "espacio de memoria" será accesible a través de su nombre (referencia). Por ejemplo si ejecuto las sentencias "CREATE OBJECT coche_uno" y "CREATE OBJECT coche_dos", tendré dos variables puntero (referencias) de tipo "coche" (de la clase coche) y cada una apunta a una dirección distinta de memoria, cada una con los valores que le asignemos.

2.3 EJEMPLO

Vamos a partir de un sencillo grupo de funciones como es el caso de un contador. Supongamos que tenemos el grupo de funciones CONTADOR con la siguiente estructura:

```
FUNCTION-POOL contador.

DATA: cont TYPE i.

FUNCTION fijar_contador.
* Interface local -> importing value (fijar_valor)
  cont = fijar_valor.
ENDFUNCTION.

FUNCTION incrementar_contador.
  ADD 1 TO cont.
```

```
ENDFUNCTION.

FUNCTION obtener_contador.
* Interface local -> exporting value (obtener_valor)
  obtener_valor = cont.
ENDFUNCTION.
```

El grupo de funciones tiene un campo de tipo entero llamado **cont** y tres módulos función, **fijar_contador**, **incrementar_contador** y **obtener_contador** que trabajan con este campo. Dos de los módulos de funciones tienen parámetro input y output. Los módulos función conforman la interface del grupo de funciones.

Un programa ABAP (p.e. un report) puede trabajar con este grupo de funciones:

```
DATA numero TYPE i VALUE 5.

CALL FUNCTION 'FIJAR_CONTADOR'
  EXPORTING
    fijar_valor = numero.

DO 3 TIMES.
  CALL FUNCTION 'INCREMENTAR_CONTADOR'.
ENDDO.

CALL FUNCTION 'OBTENER_CONTADOR'
  IMPORTING
    obtener_valor = numero.
```

Después de que esta sección del programa haya sido ejecutada, la variable numero tendrá el valor 8. El programa no puede acceder por sí mismo al campo “cont” del grupo de funciones. Las operaciones sobre

este campo están encapsuladas en el módulo de funciones. El programa sólo puede comunicarse con el grupo de funciones mediante la llamada a los módulos de funciones.

3 CLASES LOCALES

Las clases son las plantillas de los objetos. A la inversa, podemos decir que el tipo de un objeto es el mismo que el de su clase. Una clase es la descripción abstracta de un objeto. Los atributos de los objetos están definidos por los componentes de la clase (atributos, métodos y eventos), que son los que describen y controlan el comportamiento de los objetos.

3.1 CLASES LOCALES Y GLOBALES

Las clases en ABAP Objects se pueden declarar bien globalmente o bien localmente. Las clases globales se definen en el generador de clases (transacción SE24) en el ABAP Workbench, que veremos más tarde. Estas clases son almacenadas en “class-pool” en la librería de clases en el repositorio de SAP R/3. Todos los programas ABAP pueden acceder a las clases globales. Las clases locales se definen en un programa ABAP. Las clases locales y sus interfaces sólo pueden ser invocadas desde el programa en el que se han definido.

Cuando se usa una clase en un programa ABAP el sistema busca primero una clase local con el nombre especificado. Si no encuentra ninguna entonces buscará una clase global. A parte de la cuestión de la visibilidad, no hay ninguna diferencia entre usar una clase global o una clase local. Lo que si cambia sensiblemente es la manera en la que una clase local y una clase global son creadas.

3.2 DEFINICIÓN DE CLASES LOCALES

Las clases locales son el conjunto de sentencias que están entre las sentencias CLASS… y ENDCLASS. Una definición completa de una clase constará de una parte declarativa en la que se definen los componentes, y si es necesario una parte de implementación en la que se implementan estos componentes. La parte declarativa de una clase está comprendida entre las sentencias:

CLASS <nombre de la clase> DEFINITION.
…
ENDCLASS.

La parte declarativa contiene la declaración de todos los componentes de la clase (atributos, métodos y eventos). Cuando se definen clases locales, la parte declarativa pertenece a los datos globales del programa, por tanto se habrá de situar al principio del programa.

Todos los métodos que existan en la parte declarativa de una clase, deberán existir también en la parte de implementación. Ésta es la que va incluida entre las siguientes sentencias:

CLASS < nombre de la clase > IMPLEMENTATION.
…
ENDCLASS.

La parte de implementación contiene la implementación de todos los métodos de la clase. Esta parte actúa como un bloque, esto quiere decir que cualquier sección de código que no forme parte del bloque no será accesible.