LOGALI

TEORÍA

Programación orientada a objetos

SAP ABAP Programasión

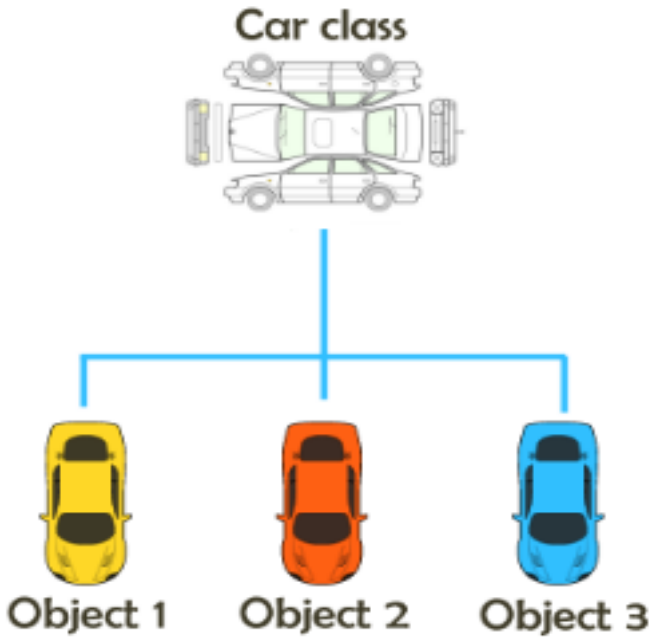
Contenido

1. Introducción

La programación orientada a objetos se basa en la programación de clases; a diferencia de la programación estructurada, que está centrada en las funciones. Una clase es un molde del que luego se pueden crear múltiples objetos, con similares características.

Una clase es una plantilla (molde), que define atributos (variables) y métodos (funciones).
La clase define los atributos y métodos comunes a los objetos de ese tipo, pero luego, cada objeto tendrá sus propios valores y compartirán las mismas funciones.
Debemos crear una clase antes de poder crear objetos (instancias) de esa clase. Al crear un objeto de una clase, decimos que hemos creado una instancia de la clase o un objeto propiamente dicho.

La posibilidad de crear diversas instancias en tiempo de ejecución utilizando el mismo contexto de programa es una de las características clave de la programación orientada a objetos.



En este ejemplo tenemos tres objetos de la misma clase, todos con distintas características.

2. Clases – Conceptos

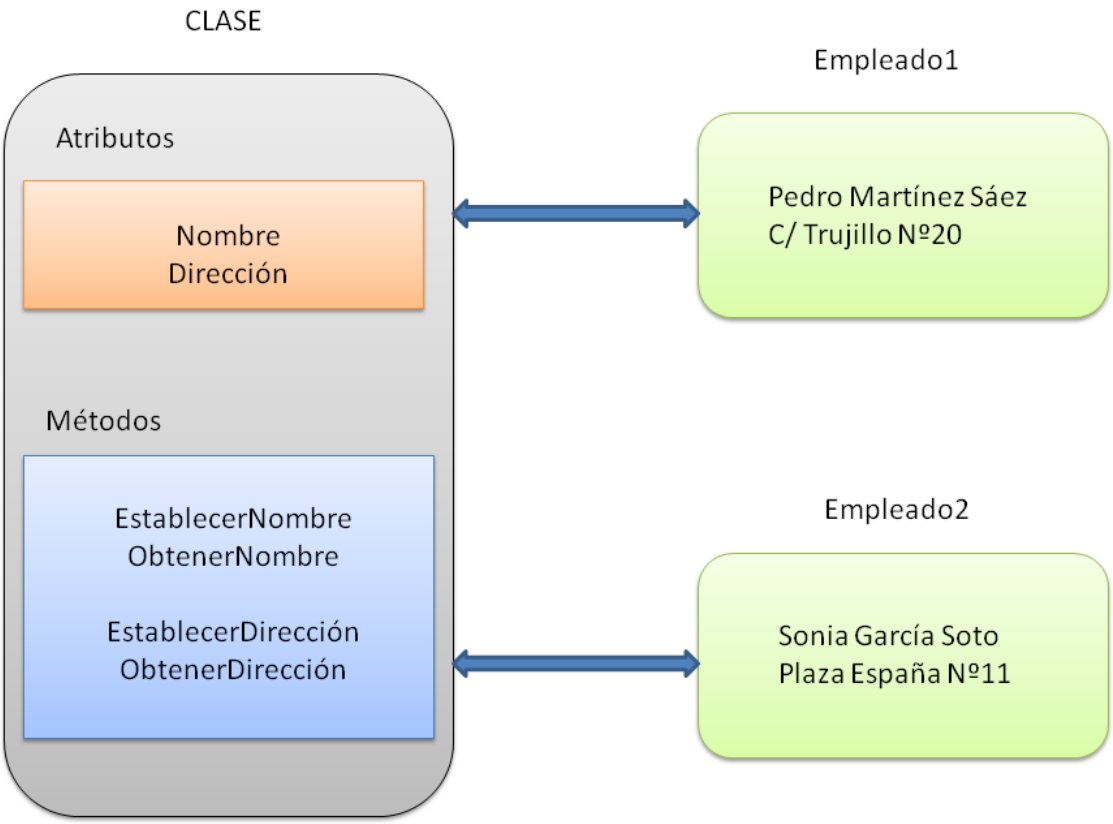
La programación orientada a objetos ve el mundo real como una colección de objetos; por ejemplo, aviones, coches y personas. Algunos de estos objetos son similares. En otras palabras, los objetos pueden describirse de la misma manera si utilizan las mismas características y muestran el mismo comportamiento.

Puede agrupar todas las características y comportamientos de estos objetos similares en una clase central. Esta clase se utiliza para describir todos los objetos derivados de ella. Por lo tanto, una clase es una descripción de una cantidad de objetos que muestra las mismas características y el mismo comportamiento.

Por ejemplo, el vehículo (marca x, … serie n), es un objeto de la clase coche. Este objeto es una instancia concreta de su clase. Por lo tanto, tiene una identidad, un estado (número de instancias características) y un comportamiento. Los conceptos de identidad y estado son diferentes entre sí.

La identidad es un atributo que distingue cada objeto de entre el resto de objetos de su clase. Dos objetos pueden tener valores de atributo idénticos pero no ser idénticos. Por ejemplo, dos tazas de café son iguales en altura, diámetro, asa y color. Aunque sus estados son completamente idénticos, se trata de dos tazas de café diferentes.

Una vez tenemos nuestra clase definida, podemos cargar en memoria (instanciar) distintos objetos de esta clase. Dado que una imagen vale más que mil palabras, veamos el siguiente diagrama.



Como podemos ver en la figura, la clase funciona como un contenedor (encapsulación) permitiendo agrupar tanto los datos (atributos) como los métodos.

También observamos que hemos creado dos objetos diferentes de la misma clase. Cada uno con sus datos particulares de nombre y dirección.

3. Tipos de clases: Globales Vs. Locales

Las clases en ABAP Objects pueden ser declaradas globalmente o localmente. Así como existe una manera de definir un Function Group o un Function Module desde el ABAP Workbench para ser usados en la programación ABAP tradicional, también es posible definir las clases globales y las interfaces en el ABAP Workbench mediante el Class Builder (Transacción SE24). Estas clases son almacenadas centralmente en pools de clases en la biblioteca de clases dentro del repositorio de SAP.

Todos los programas ABAP en un sistema SAP pueden acceder a las clases globales. Las clases locales son las que se definen dentro de un programa ABAP. De esta manera, las interfaces y las clases locales pueden ser usadas solamente en el programa en donde están definidas. Cuando se usa una clase en un programa ABAP, el sistema primero busca una clase local con el nombre especificado. Si no se encuentra ninguna, busca una clase global.

4. Definición de una clase local

La definición completa de una clase en ABAP Objects consiste en una sección de **declaración** y otra de **implementación**, ambas iniciadas con la sentencia **CLASS**.

Sentencia para la **declaración** de una clase local:

CLASS <nombre_clase> DEFINITION.
…
ENDCLASS.

Sentencia para la **implementación** de una clase local:

CLASS < **nombre_clase**> IMPLEMENTATION.
…
ENDCLASS.

Ejemplo:

```
CLASS empleado DEFINITION.

  PUBLIC SECTION.

    DATA: nombre TYPE string.

    METHODS: establecer_nombre IMPORTING i_nombre TYPE string,
             obtener_nombre    EXPORTING e_nombre TYPE string.
ENDCLASS.

CLASS empleado IMPLEMENTATION.

  METHOD establecer_nombre.
```

```
    nombre = i_nombre.

  ENDMETHOD.

  METHOD obtener_nombre.

    e_nombre = nombre.

  ENDMETHOD.
ENDCLASS.
```

5. Utilización de objetos

5.1. Objetos

Los objetos son instancias de las clases. Cada objeto tiene una identidad propia y tiene sus propios atributos. Todos los objetos transitorios residen en el contexto de una sesión interna (área de memoria de un programa ABAP). Una clase puede tener un número indefinido de objetos (instancias).

5.2. Referencias a objeto

Las referencias a objeto se usan para acceder a un objeto desde un programa ABAP. Las referencias a objeto son punteros a los objetos. En ABAP los objetos están siempre contenidos en variables referenciadas.

Las variables referenciadas o bien contienen el valor 'initial' o bien contienen la Referencia a un objeto ya existente. La identidad de un objeto depende de su referencia. Una variable referenciada que apunta a un objeto es la que conoce la identidad del objeto. Los usuarios no pueden acceder a la identidad del objeto directamente. Las variables referenciadas en ABAP son tratadas como cualquier otro objeto de datos elemental. Esto quiere decir que una variable referenciada puede contener una tabla interna o una estructura.

ABAP contiene un tipo de datos predefinido para las referencias, comparable a los tipos de datos para las estructuras o para las tablas internas. El tipo de datos completo no está definido hasta la declaración en el programa ABAP. El tipo de datos de la variable referenciada determina cómo el programa actúa con su valor, o sea, con la referencia al objeto. Hay

dos tipos principales de referencias, la referencia a clases y la referencia a interfaces (se verá más adelante).
Las referencias a clases se definen usando la siguiente adición:

… TYPE REF TO.

Esta adición se usa en las sentencias TYPES o DATA. Una variable referenciada de este tipo se llama variable referenciada a clase o referencia a clase simplemente. Una referencia a clase permite al usuario crear una instancia, o sea, un objeto de la clase y acceder a los componentes visibles.

Ejemplo:

```
DATA: gcl_empleado_1 TYPE REF TO empleado,
      gcl_empleado_2 TYPE REF TO empleado.
```

¿Cómo crear objetos?

Antes de crear un objeto de una clase es necesario declarar una variable referenciada con la referencia a la clase. Una vez que se ha declarado la referencia a la clase, se puede crear el objeto usando la sentencia CREATE OBJECT. Esta sentencia crea una instancia de la clase, y la variable referenciada contiene la referencia al objeto.

```
CREATE OBJECT: gl_empleado_1,
               gl_empleado_2.
```

6. Encapsulación

Encapsulamiento: Es el término que define formalmente la fusión dentro del mismo objeto de sus datos y métodos (comportamiento). Los objetos son tratados como una “caja negra”, de manera que sólo sus métodos pueden actuar sobre sus datos, brindando independencia en la implementación de los mismos.

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.

Podemos entender el encapsulamiento como la visibilidad permitida sobre el objeto. La parte declarativa de una clase se divide en tres áreas de distinta visibilidad:

```
CLASS empleado DEFINITION.

      PUBLIC SECTION.
      .........
      PROTECTED SECTION.
      .........
      PRIVATE SECTION.

        .........

ENDCLASS.
```

1. Sección publica – PUBLIC SECTION.
2. Sección protegida – PROTECTED SECTION.
3. Sección privada – PRIVATE SECTION.

Estas tres áreas definen la visibilidad externa de los componentes de la clase, esto es, la interface entre la clase y el usuario. Cada componente de una clase ha de ser asignado a una de estas tres secciones:

- PUBLIC SECTION – todos los componentes declarados en la sección pública son accesibles para todos los usuarios de la clase y para todos los métodos de la clase y de cualquier clase que herede de ella.

- PROTECTED SECTION – todos los componentes declarados en la sección protegida son accesibles para todos los métodos de la clase y de las clases que heredan de ella. Los componentes protegidos conforman la interface entre una clase y todas sus subclases.

- PRIVATE SECTION Los componentes declarados en la sección privada son sólo visibles en los métodos de la misma clase. Los componentes privados no forman parte de la interface externa de la clase.

Las tres áreas de visibilidad son la base de una de las más importantes características de la orientación a objetos, la encapsulación. Cuando se

define una clase hay que tener mucho cuidado en el diseño de los componentes públicos, intentando declarar tan pocos como sea posible. Los componentes públicos de las clases globales no pueden ser cambiados una vez que se ha liberado la clase. Por ejemplo, los atributos públicos son visibles externamente, y forman parte de la interface entre un objeto y sus usuarios. Si se quiere encapsular el estado de un objeto completamente no se tiene que declarar ningún atributo público. Además de definir la visibilidad de un atributo, se puede proteger también de los cambios usando la adición **READ-ONLY**.

7. Atributos de instancia y atributos estáticos

Los atributos son los campos de datos internos de una clase y pueden tener cualquier tipo de datos ABAP. El estado de un objeto viene determinado por el contenido de sus atributos. Un tipo de atributos son las variables referenciadas. Estas variables permiten crear y acceder a los objetos, de manera que si se definen en una clase permiten acceder a otros objetos desde dentro de la clase.

7.1. Tipos de atributos

- **Atributos dependientes de instancia:** el contenido de estos atributos es específico de cada objeto. Se declaran usando la sentencia **DATA**.

- **Atributos estáticos** – el contenido de los atributos estáticos define el estado de la clase y es válido para todas las instancias de la clase. Los atributos estáticos existen sólo una vez para la clase. Se declaran usando la sentencia **CLASS-DATA.** Son accesibles desde todo el entorno de ejecución de la clase. Todos los objetos de una clase pueden acceder a sus atributos estáticos. Si se cambia un atributo estático en un objeto, el cambio es visible en todos los demás objetos de la clase.

Los tipos de atributos de clase son los siguientes:

- Elemental
- Estructurado

- Tipo de tabla

Los atributos pueden consistir en tipos de datos (locales o globales) o tipos de referencia.

7.2. Definición de atributos, tipos y constantes

En las sentencias DATA de las clases, solo puede utilizar el suplemento TYPE para hacer referencia a los tipos de datos.

El suplemento LIKE solo está permitido para objetos de datos locales o campos SY (por ejemplo SY-DATE, SY-UNAME, etc.).

El suplemento READ-ONLY indica si un atributo público declarado con DATA puede leerse desde el exterior. Sin embargo, el atributo solo puede modificarse mediante métodos en la misma clase. Actualmente, puede utilizar el suplemento READ-ONLY en la sección de visibilidad pública (PUBLIC SECTION) de una declaración de clase o en una definición de interfaz.

Con TYPE REF TO, puede indicar un atributo como una referencia.

La sentencia CONSTANTS se utiliza en la definición de clase para definir los objetos de datos que tienen un valor constante.

Si utiliza la sentencia TYPES en la definición de clase, declara un tipo local específico a la clase local. Puede crear un tipo local para que lo utilicen uno u más atributos dentro de la misma clase.

8. Métodos estáticos y métodos de instancia

8.1. Declaración de métodos

Los métodos se pueden declarar bien en la parte declarativa de una clase o bien en una interface. Para declarar métodos dependientes de instancia se usa la siguiente sentencia:

```
METHODS <meth>
IMPORTING.. [VALUE(]<ii>[)]   TYPE type [OPTIONAL]..
EXPORTING.. [VALUE(]<ei>[)] TYPE type [OPTIONAL]..
CHANGING.. [VALUE(]<ci>[)]   TYPE type [OPTIONAL]..
RETURNING VALUE(<r>)
EXCEPTIONS.. <ei>..
```

Para declarar métodos estáticos se usa se usa la siguiente sentencia:

```
CLASS-METHODS <meth>..
```

Ambas sentencias tienen la misma sintaxis. Cuando se declara un método se puede definir su interface de parámetros usando las adiciones:

- IMPORTING
- EXPORTING
- CHANGING
- RETURNING

Estas adiciones definen los parámetros de entrada, de salida, de entrada/salida y el código que devuelve el método. También definen si los parámetros se pasan por referencia o por valor (VALUE), su tipo (TYPE), o si es opcional o por defecto (OPTIONAL, DEFAULT). Al contrario que en los módulos de funciones, el modo por defecto de pasar parámetros a un método es por referencia. Para pasar un parámetro por valor es necesario especificar explícitamente la adición VALUE.

8.2. Implementación de métodos

Los métodos se implementan en la parte de implementación de la clase con las sentencias:

```
METHOD <meth>.
...
ENDMETHOD.
```

Al implementar un método no hay que especificar los parámetros de la interface del método, ya que están definidos en la declaración del método. La interface de parámetros de un método se comporta dentro de la implementación del método como variables locales. Se pueden definir variables locales adicionales dentro del método usando la sentencia DATA.

Al igual que en los módulos de funciones, se pueden usar las sentencias RAISE <exception> y MESSAGE RAISING para controlar los errores.

Cuando se implementa un método estático hay que tener en cuenta que sólo puede trabajar con los atributos estáticos de la clase. Los métodos dependientes de instancia pueden trabajar tanto con atributos estáticos como con atributos dependientes de instancia.

9. Constructor de instancia y constructor estático

Los constructores son un tipo especial de métodos que no pueden ser llamados con la sentencia CALL METHOD. Estos métodos son llamados automáticamente por el sistema para fijar el estado inicial de un nuevo objeto o clase. Hay dos tipos de constructores, los dependientes de instancia y los estáticos o independientes de instancia. Los constructores son métodos con un nombre predefinido. Para usarlos deben ser declarados explícitamente en la clase.

El constructor dependiente de instancia de una clase es un método que se llama CONSTRUCTOR. Se declara en la sección pública de la siguiente manera:

```
METHODS CONSTRUCTOR
IMPORTING.. [VALUE(]<ii>[)] TYPE type [OPTIONAL].
EXCEPTIONS.. <ei>.
```

Se implementa en la parte de implementación de la misma manera que cualquier otro método. El sistema llama al constructor dependiente de instancia una vez para cada instancia de la clase, justo después de que el objeto haya sido creado mediante la sentencia CREATE OBJECT.

Se le pueden pasar parámetros de entrada y controlar sus errores usando las adiciones EXPORTING y EXCEPTIONS en la sentencia CREATE OBJECT.

El constructor estático de una clase es el método estático predefinido CLASS_CONSTRUCTOR. Se declara en la sección pública de la siguiente manera:

```
CLASS-METHODS CLASS_CONSTRUCTOR.
```

Se implementa como cualquier otro método. El sistema llama al constructor estático una vez para cada clase, justo antes de que la clase se utilice por primera vez. Debido a esto, el constructor estático no puede acceder a los componentes de la propia clase.

El sistema llama el constructor estático de manera automática antes de que se accede a la clase por primera vez y antes de la primera ejecución de las siguientes acciones:

- Cuando se crea una instancia de la clase (CREATE OBJECT).
- Cuando se accede a un atributo estático de la clase.
- Cuando se llama un método estático de la clase.
- Cuando se registra un método de programa de control de eventos para un evento en la Clase

Cuando define los constructores estáticos, siempre debe tener en cuenta los puntos siguientes:

- Cada clase no tiene más que un constructor estático.
- El constructor estático debe definirse en el área pública.
- La firma del constructor no puede tener parámetros de importación ni excepciones.
- El constructor estático no puede llamarse de forma explícita.