LOGALI

TEORÍA

Interfaces y clases abstractas

SAP ABAP Programación Orientada a Objetos

Contenido

1. Interfaces

1.1. Concepto

Las interfaces difieren de la herencia habitual en sus áreas de uso. Sin embargo, en cuanto a la programación, prácticamente no hay diferencias entre las interfaces y la herencia normal. Las interfaces pueden verse como clases superiores que no pueden instanciarse, no contienen implementaciones y solo tienen componentes públicos. Se puede simular la herencia múltiple utilizando las interfaces.

En los objetos ABAP, las interfaces se utilizan principalmente para definir protocolos de interfaces uniformes para los servicios. Las distintas clases pueden implementar estos servicios de diferentes maneras, pero debe mantener la misma semántica. De esta forma, las interfaces no contienen implementaciones.

En los objetos ABAP, los mismos componentes pueden definirse generalmente en interfaces y clases. Para reconocer las diferencias semánticas de la herencia normal, puede concentrarse en los casos de uso siguientes. Por ejemplo, desea permitir que múltiples clases implementen un servicio de diferentes maneras, pero utilizando los mismos nombres de método y una firma uniforme. Con la herencia normal, un método de este tipo se define en la clase superior compartida. Sin embargo, si no se puede modelar una clase superior de manera adecuada para la herencia, se deberá definir una interfaz y después el método en la interfaz. Por lo tanto, este caso se puede comparar con una relación de generalización dentro de una clase superior.

En comparación con la herencia normal la distribución de roles en interfaces es a veces diferente. El usuario generalmente define las interfaces. En estas interfaces, el usuario describe, tanto técnica como semánticamente, los servicios que desea recibir del proveedor. Cada clase puede decidir ahora por sí misma si sirve a la interfaz, es decir, si realmente ofrece los servicios definidos en la interfaz. Por ello, este caso es similar a una relación de especialización con una subclase.

2. Definir interfaces

Para definir una interfaz en ABAP utilizamos la palabra clase INTERFACE y cerramos la definición con ENDINTERFACE. Dentro de la interfaz podemos definir componentes (atributos, métodos, tipos).

```
interface zif_lab_02_customer
  public .

  types: begin of ty_cust_address,
           first_name  type string,
           last_name   type string,
         end of ty_cust_address.

  methods get_customer importing iv_customer_id     type string
                       returning value(rs_customer) type ty_cust_address.

endinterface.
```

2.1. Implementación de interfaces

Las interfaces se pueden implementar por clases o por interfaces. Para implementar la interfaz se utiliza la palabra clave INTERFACES. Por las clases se tienen que implementar dentro de la sección pública porque todos los componentes de la interfaz son públicos.

2.2. Implementación de una interfaz por la clase

ZCL_LAB_26_FLIGHTS

```
1  class zcl_lab_26_flights definition
2    public
3    final
4    create public .
5
6    public section.
7      interfaces zif_lab_01_flight.
8      interfaces zif_lab_02_customer.
```

2.3. Implementación de una interfaz por otra interfaz

```
INTERFACE if_1
  INTERFACES if_2.
ENDINTERFACE.
```

2.4. Acceso a los componentes de la interfaz

El acceso a los componentes de la interfaz se puede hacer solo mediante una referencia de un objeto que ha implementado la interfaz. Utiliza el operador de resolución de interfaz (~) para acceder a los componentes de interfaz de la parte de implementación de la clase. De manera alternativa, se pueden utilizar los nombres de alias definidos en la clase de implementación para los componentes de interfaz. Incluso si los componentes compartidos de las clases de implementación se transfieren posteriormente a la interfaz, no será necesario adaptar el acceso a estos componentes.

```
interface_name~component_name
```

2.5. Implementación de múltiples interfaces

Una clase o una interfaz puede implementar múltiples interfaces, no existe un límite de interfaces que se pueden implementar. Se utiliza la misma palabra clave INTERFACES y se pasa el listado con todas las interfaces que se implementan.

```
INTERFACES: if_1, if_2, if_3……..if_n.
```

La clase que implementa una interfaz que a su vez ha implementado otra interfaz tiene la obligación de implementar todos los métodos de las dos interfaces.

Por ejemplo:

```
INTERFACE if_1.

  METHODS m_1.

ENDINTERFACE.
```

Y la IF_2 implementa la IF_1

```
INTERFACE if_1.

  INTERFACES if_2.
METHODS m_2.

ENDINTERFACE.
```

Ahora si una clase implementa la interfaz IF_2 tiene la obligación de implementar todos los métodos declarados en las interfaces IF_1 y IF_2, en este caso los métodos M_1 y M_2.

```
CLASS nombre_clase DEFINITION.

  PUBLIC SECTION.
    INTERFACES: if_2.

ENDCLASS.

CLASS nombre_clase IMPLEMENTATION.

  METHOD if_1~m_1.

  ENDMETHOD.

  METHOD if_2~m_2.

  ENDMETHOD.

ENDCLASS.
```

3. Aliases

Para simplificar el acceso a los componentes de interfaz, puede utilizar nombres alias. Estos solo pueden aparecer en la parte de definición de una clase o en la definición de interfaz. El uso de nombres alias está sujeto a la restricción de visibilidad de la clase de definición. A continuación se define un alias para un método de interfaz:

```
ALIASES a_1 FOR if_1~method_1.
```

El método de interfaz if_1~method_1 puede llamarse con la forma corta ref->a_1.

4. Clases abstractas

4.1. Conceptos

La clase abstracta contiene definición e implementación, pero no puede instanciarse. Utilice el suplemento ABSTRACT en la sentencia CLASS para crear una clase abstracta. Las clases superiores son un uso típico de las clases abstractas, ya que no se instancias por sí solas, pero sus subclases sí.

En una clase abstracta, se pueden definir métodos abstractos (entre otras cosas). Esto significa que el método abstracto no puede implementarse en esa clase. En lugar de ello, se implementa en una subclase de la clase. Si la subclase de esa clase no es abstracta, los métodos abstractos deben redefinirse e implementarse en la subclase por primera vez.

4.2. Creación de clase abstracta

Utilizamos la palabra clase ABSTRACT en la definición de la clase.

```
CLASS nombre_clase DEFINITION ABSTRACT.

  PUBLIC SECTION.

    METHODS metodo_abstracto ABSTRACT.

ENDCLASS.
```

La clase abstracta debe contener al menos un método abstracto. Se utiliza la misma palabra clave ABSTRACT en la definición de método.

4.3. Consideraciones

- No se puede definir un método abstracto como FINAL, porque no se puede redefinir y el propósito de los métodos abstractos es para que se redefina en las subclases.
- Si una clase abstracta hereda de otra clase abstracta no tiene la obligación de implementar los métodos abstractos de la clase superior. La primera subclase que no es abstracta del árbol jerárquico tiene la obligación de implementar todos los métodos de todas las clases superiores abstractas.

5. Diferencias entre interfaces y clases abstractas

- **Herencia múltiple**: Podemos lograr múltiples herencias usando interfaces. Como ABAP no admite más de una clase Padre (Clase Superior), sólo puede tener una clase abstracta como clase Padre.
- **Nueva funcionalidad**: Si añadimos un nuevo método en la interfaz, todas las clases que han implementado la interfaz tienen que implementar este método. Si no implementamos el método va resultar un error en tiempo de ejecución RUN – Time error. Para la clase abstracta, si añadimos un método no abstracto, no es necesario volver a implementar el método en cada clase heredada.
- **Comportamiento por defecto**: Podemos tener un comportamiento predeterminado de un método en la clase abstracta. No podemos tener

ningún comportamiento predeterminado en la interfaz, ya que sólo contiene la firma de los métodos.

- **Visibilidad**: Todos los componentes de la interfaz son públicos por defecto. Para la clase abstracta, podemos establecer la visibilidad de cada componente.